Testeur Certifié

Syllabus Niveau Fondation

Version 2018 FR

International Software Testing Qualifications Board

Comité Français des Tests Logiciels

Historique des modifications

Version	Date	Remarques
CFTL 2018FR	31-Août-2018	Version française avec corrections mineures de traduction
CFTL 2018FR	30-Juin-2018	Version française
ISTQB 2018	27-Avr-2018	Version finale
ISTQB 2018	12-Fév-2018	Version Beta
ISTQB 2018	19-Janv-2018	Version Alpha 3.0
ISTQB 2018	15-Janv-2018	Version Alpha 2.9 - Revue interne (cross review) et édition technique
ISTQB 2018	9-Déc-2017	Version Alpha 2.5
ISTQB 2018	22-Nov-2017	Version Alpha 2.0
ISTQB 2018	12-Juin-2017	Version Alpha
CFTL 2011FR	3-Nov-2011	Testeur Certifié Niveau Fondation Mise à jour – voir Annexe C - Notes de la version 1
CFTL 2010FR	1-Sept-2010	Testeur Certifié Niveau Fondation Mise à jour – voir Annexe C - Notes de la version
ISTQB 2010	30-Mars-2010	Certified Tester Foundation Level Syllabus Maintenance Release – voir Annexe C - Notes de la version
ISTQB 2007	01-Mai-2007	Certified Tester Foundation Level Syllabus Maintenance Release – voir Appendix E – Release Notes Syllabus 2007
CFTL 2005FR	01-Juillet-2005	Testeur Certifié Niveau Fondation
ISTQB 2005	01-Juillet-2005	Certified Tester Foundation Level Syllabus
ASQF V2.2	Juillet-2003	ASQF Syllabus Foundation Level Version 2.2 “Lehrplan Grundlagen des Software-testens“
ISEB V2.0	25-Février-1999	ISEB Software Testing Foundation Syllabus V2.0 25 February 1999

Table des matières

Remerciements

Ce document a été approuvé formellement comme version officielle du syllabus Testeur Certifié Niveau Fondation, version 2018, par l'assemblée générale de l'ISTQB (4 juin 2018).

Il a été produit par une équipe de l'ISTQB composée de : Klaus Olsen (responsable), Tauhida Parveen (co-responsable), Rex Black (chef de projet), Debra Friedenberg, Judy McKay, Meile Posthuma, Hans Schaefer, Radoslaw Smilgin, Mike Smith, Steve Toms, Stephanie Ulrich, Marie Walsh, et Eshraka Zakaria.

L'équipe remercie Rex Black et Dorothy Graham pour l'édition technique du document, et l'équipe de revue, l'équipe de revue croisée et les comités nationaux pour leurs apports et propositions d'amélioration.

Les personnes suivantes ont contribué aux revues, commentaires et échanges sur ce syllabus : Tom

Adams, Tobias Ahlgren, Xu Aiguo, Chris Van Bael, Katalin Balla, Graham Bath, Gualtiero Bazzana, Arne Becher, Veronica Belcher, Lars Hilmar Bjørstrup, Ralf Bongard, Armin Born, Robert Bornelind, Mette Bruhn-Pedersen, Geza Bujdoso, Earl Burba, Filipe Carlos, Young Jae Choi, Greg Collina, Alessandro Collino, Cui Zhe, Taz Daughtrey, Matthias Daigl, Wim Decoutere, Frans Dijkman, Klaudia Dussa-Zieger, Yonit Elbaz, Ofer Feldman, Mark Fewster, Florian Fieber, David Frei, Debra Friedenberg, Conrad Fujimoto, Pooja Gautam, Thorsten Geiselhart, Chen Geng, Christian Alexander Graf, Dorothy Graham, Michel Grandjean, Richard Green, Attila Gyuri, Jon Hagar, Kobi Halperin, Matthias Hamburg, Zsolt Hargitai, Satoshi Hasegawa, Berit Hatten, Wang Hongwei, Tamás Horváth, Leanne Howard, Chinthaka Indikadahena, J. Jayapradeep, Kari Kakkonen, Gábor Kapros, Beata Karpinska, Karl Kemminger, Kwanho Kim, Seonjoon Kim, Cecilia Kjellman, Johan Klintin, Corne Kruger, Gerard Kruijff, Peter Kunit, Hyeyong Kwon, Bruno Legeard, Thomas Letzkus, Alon Linetzki, Balder Lingegård, Tilo Linz, Hongbiao Liu, Claire Lohr, Ine Lutterman, Marek Majernik, Rik Marselis, Romanos Matthaios, Judy McKay, Fergus McLachlan, Dénes Medzihradszky, Stefan Merkel, Armin Metzger, Don Mills, Gary Mogyorodi, Ninna Morin, Ingvar Nordström, Adam Novak, Avi Ofer, Magnus C Ohlsson, Joel Oliviera, Monika Stocklein Olsen, Kenji Onishi, Francisca Cano Ortiz, Gitte Ottosen, Tuula Pääkkönen, Ana Paiva, Tal Pe'er, Helmut Pichler, Michaël Pilaeten, Horst Pohlmann, Andrew Pollner, Meile Posthuma, Vitalijs Puiso, Salvatore Reale, Stuart Reid, Ralf Reissing, Shark Ren, Miroslav Renda, Randy Rice, Adam Roman, Jan Sabak, Hans Schaefer, Ina Schieferdecker, Franz Schiller, Jianxiong Shen, Klaus Skafte, Mike Smith, Cristina Sobrero, Marco Sogliani, Murian Song, Emilio Soresi, Helder Sousa, Michael Sowers, Michael Stahl, Lucjan Stapp, Li Suyuan, Toby Thompson, Steve Toms, Sagi Traybel, Sabine Uhde, Stephanie Ulrich, Philippos Vakalakis, Erik van Veenendaal, Marianne Vesterdal, Ernst von Düring, Salinda Wickramasinghe, Marie Walsh, Søren Wassard, Hans Weiberg, Paul Weymouth, Hyungjin Yoon, John Young, Surong Yuan, Ester Zabar, et Karolina Zmitrowicz.

Groupe de travail International Software Testing Qualifications Board Niveau Fondation (Edition 2018): Klaus Olsen (chair), Tauhida Parveen (vice chair), Rex Black (project manager), Dani Almog, Debra Friedenberg, Rashed Karim, Johan Klintin, Vipul Kocher, Corne Kruger, Sunny Kwon, Judy McKay, Thomas Müller, Igal Levi, Ebbe Munk, Kenji Onishi, Meile Posthuma, Eric Riou du Cosquer, Hans Schaefer, Radoslaw Smilgin, Mike Smith, Steve Toms, Stephanie Ulrich, Marie Walsh, Eshraka Zakaria, et Stevan Zivanovic. L'équipe principale remercie l'équipe de revue et tous les Comités Nationaux pour leurs suggestions.

Groupe de travail International Software Testing Qualifications Board Niveau Fondation (Edition 2011): Thomas Müller (chair), Debra Friedenberg. Le groupe de travail remercie l'équipe de revue (Dan Almog, Armin Beer, Rex Black, Julie Gardiner, Judy McKay, Tuula Pääkkönen, Eric Riou du Cosquer, Hans Schaefer, Stephanie Ulrich, Erik van Veenendaal) et tous les Comités Nationaux pour leurs suggestions

pour la version actuelle du syllabus. L'équipe des auteurs du Syllabus remercie l'équipe des questions d'examen et tous les Comités Nationaux pour leurs suggestions.

Groupe de travail International Software Testing Qualifications Board Niveau Fondation (Edition 2010): Thomas Müller (responsable), Rahul Verma, Martin Klonk et Armin Beer. Le groupe de travail remercie l'équipe de revue (Rex Black, Mette Bruhn-Pederson, Debra Friedenberg, Klaus Olsen, Tuula Pääkkönen, Meile Posthuma, Hans Schaefer, Stephanie Ulrich, Pete Williams, Erik van Veenendaal) et tous les Comités Nationaux pour leurs suggestions pour la version actuelle du syllabus.

Groupe de travail International Software Testing Qualifications Board Niveau Fondation (Edition 2007): Thomas Müller (responsable), Dorothy Graham, Debra Friedenberg, et Erik van Veendendaal Le groupe de travail remercie l'équipe de revue (Hans Schaefer, Stephanie Ulrich, Meile Posthuma, Anders Pettersson, et Wonil Kwon) et tous les Comités Nationaux pour leurs suggestions.

Groupe de travail International Software Testing Qualifications Board Niveau Fondation (Edition 2005): Thomas Müller (responsable), Rex Black, Sigrid Eldh, Dorothy Graham, Klaus Olsen, Maaret Pyhäjärvi, Geoff Thompson, Erik van Veenendaal et l'équipe de revue ainsi que tous les Comités Nationaux pour leurs suggestions.

0 Introduction

0.1 Objectif de ce Syllabus

Ce syllabus forme la base du niveau Fondation de la qualification Internationale en tests de logiciels. L'International Software Testing Qualifications Board (ISTQB) et le Comité Français des Tests Logiciels (ci-après appelé CFTL) fournissent ce syllabus pour le propos suivant :

1. pour les membres de l'ISTQB, afin de traduire dans leur langue locale et d'accréditer les fournisseurs de formation. Les Comités Nationaux peuvent adapter ce syllabus à leurs besoins linguistiques particuliers et ajouter des références pour l'adapter à leurs publications locales.
2. pour les organismes de certification, afin de produire les questions d'examen dans leur langue locale en fonction des objectifs d'apprentissage pour ce syllabus.
3. pour les organismes de formation, afin de produire le matériel de formation et déterminer les méthodes pédagogiques appropriées.
4. pour les candidats à la certification, pour se préparer à l'examen de certification (dans le cadre d'un cours de formation ou indépendamment).
5. pour la communauté internationale de l'ingénierie des logiciels et des systèmes, pour faire progresser les pratiques professionnelles en tests de logiciels et de systèmes, et comme base pour des livres articles.

L'ISTQB peut permettre à d'autres entités d'utiliser ce syllabus à d'autres fins, à condition que l'autorisation en soit demandée et accordée de façon écrite au préalable par l'ISTQB.

0.2 Le niveau fondation pour les testeurs de Logiciels certifiés

La qualification de niveau fondation vise toutes les personnes impliquées dans les tests de logiciels. Ceci inclut les personnes ayant des rôles de testeurs, analystes de tests, ingénieurs de tests, consultants en tests, Test Manager, testeurs en phase d'acceptation utilisateur et développeurs de logiciels. Cette qualification de niveau Fondation est aussi appropriée pour toute personne souhaitant une compréhension de base des tests de logiciels, tels que les responsables produits, chefs de projets, responsables qualité, responsables de développements logiciels, analystes métier, directeurs informatique et consultants en management. Les possesseurs du Certificat Fondation pourront poursuivre leur qualification en tests logiciels afin d'atteindre un niveau de certification plus élevé.

Le document ISTQB / CFTL de présentation de la certification Fondation en version 2018 est un document séparé qui intègre les informations suivantes :

- Compétences attestées par la certification des testeurs au niveau fondation
- Matrice montrant la traçabilité entre les compétences attestées et les objectifs d'apprentissage
- Résumé de ce syllabus

0.3 Objectif d'apprentissage et niveaux de connaissance

Les objectifs d'apprentissage visent l'obtention des compétences attestées et sont utilisés pour créer les examens pour la certification.

De manière générale, tous les contenus de ce syllabus peuvent faire l'objet de questions d'examen au niveau K1, à l'exception de l'introduction et des annexes. C'est-à-dire qu'on peut demander au candidat de reconnaître ou de se souvenir d'un mot-clé ou d'un concept mentionné dans l'un des six chapitres du présent document. Les niveaux de connaissance des objectifs d'apprentissage sont fournis au début de chaque chapitre et classés de la façon suivante :

- K1 : se souvenir
- K2 : comprendre
- K3 : appliquer

De plus amples détails et des exemples d'objectifs d'apprentissage sont donnés en Annexe B.

Tous les termes listés sous la rubrique "termes" après les titres de chapitre doivent être retenus (K1), même s'ils ne sont pas explicitement mentionnés dans les objectifs d'apprentissage.

0.4 L'examen de certification au niveau fondation

L'examen pour l'obtention du certificat fondation sera basé sur ce syllabus. Les réponses aux questions d'examen peuvent requérir l'utilisation d'informations contenues dans plus d'une section de ce syllabus. Toutes les sections de ce syllabus peuvent donner lieu à des questions d'examen à l'exception de cette introduction et des annexes. Des normes, livres et autres syllabus de l'ISTQB sont indiqués comme références, mais leur contenu ne fait pas l'objet de questions d'examen, au-delà de ce qui est résumé dans ce syllabus lui-même comme les normes, livres et autres syllabus de l'ISTQB.

Le format de l'examen est un questionnaire à choix multiples. L'examen est structuré en 40 questions. Pour réussir cet examen, il faut obtenir au moins 65% de réponses justes (c'est-à-dire au moins 26 réponses justes).

Les examens peuvent faire partie d'une formation accréditée ou être passés indépendamment (par.ex. dans un centre d'examen ou lors d'un examen public). La participation à une formation accréditée n'est pas un prérequis au passage de l'examen.

0.5 Accréditation

Les organismes de formations dont le contenu du cours suit ce syllabus peuvent être accrédités par un Comité National de l'ISTQB. Les directives d'accréditation doivent être obtenues auprès de l'organisme ou du Comité effectuant l'accréditation. Un cours accrédité est reconnu comme se conformant à ce syllabus, et est autorisé à inclure un examen ISTQB – CFTL comme partie du cours.

0.6 Niveau de détail

Le niveau de détail dans ce syllabus permet un enseignement et des examens compatibles internationalement. Pour atteindre cet objectif, le syllabus contient :

- Des objectifs généraux d'instruction, décrivant les intentions du niveau fondation
- Une liste de termes desquels les étudiants doivent se souvenir
- Des objectifs d'apprentissage pour chaque domaine de connaissance, décrivant les résultats cognitifs d'enseignements à acquérir
- Une description des concepts clé, incluant des références à des sources comme des normes ou de la littérature reconnue

Le contenu du syllabus n'est pas une description de l'ensemble du domaine de connaissance en tests logiciels ; il reflète le niveau de détail devant être couvert par les formations du niveau fondation. Il se concentre sur les concepts et les techniques de test qui peuvent s'appliquer à tous les projets logiciels, y compris les projets Agiles. Ce syllabus ne contient pas d'objectifs d'apprentissage spécifiques liés à un cycle de développement logiciel ou à une méthode particulière, mais il indique comment ces concepts s'appliquent aux projets Agiles, à d'autres types de cycles de vie itératifs et incrémentaux, et aux cycles de vie séquentiels.

0.7 Organisation du syllabus

Ce syllabus comprend six chapitres sur lesquels porte l'examen. Le titre principal de chaque chapitre spécifie la durée prévue pour traiter ce chapitre, cette durée n'est pas détaillée au niveau des sections du chapitre. Pour une formation accréditée, le programme exige un minimum de 16,75 heures d'enseignement, réparties entre les six chapitres suivants :

- Chapitre 1 : Fondamentaux des tests (175 minutes)
- Chapitre 2 : Tester pendant le cycle de vie du développement logiciel (100 minutes)
- Chapitre 3 : Tests statiques (135 minutes)
- Chapitre 4 : Techniques de test (330 minutes)
- Chapitre 5 : Gestion des tests (225 minutes)
- Chapitre 6 : Outils de support aux tests (40 minutes)

1 Fondamentaux des tests — 175 minutes

Termes

couverture, débogage, défaut, erreur, défaillance, qualité, assurance qualité, cause racine, analyse de test, base de test, cas de test, clôture des tests, condition de test, contrôle des tests, données de test, conception des tests, exécution des tests, planning de l'exécution des tests, implémentation des tests, pilotage des tests, objet de test, objectif de test, oracle de test, planification des tests, procédure de test, suite de test, test, testware, traçabilité, validation, vérification

Objectifs d'apprentissage pour le chapitre Fondamentaux des Test :

1.1 Que sont les tests ?

FL-1.1.1 (K1) Identifier les objectifs habituels des tests

FL-1.1.2 (K2) Faire la différence entre tester et déboguer

1.2 Pourquoi les tests sont-ils nécessaires ?

FL-1.2.1 (K2) Donner des exemples montrant la nécessité des tests

FL-1.2.2 (K2) Expliquer la relation entre les tests et l'assurance qualité et donner des exemples montrant comment les tests contribuent à une amélioration de la qualité

FL-1.2.3 (K2) Faire la différence entre erreur, défaut et défaillance

FL-1.2.4 (K2) Faire la différence entre la cause racine d'un défaut et ses effets

1.3 Les 7 Principes généraux des tests

FL-1.3.1 (K2) Expliquer les 7 principes généraux des tests

1.4 Processus de test

FL-1.4.1 (K2 Expliquer l'impact du contexte sur le processus de test

FL-1.4.2 (K2) Décrire les activités de test et les tâches associées, au sein du processus de test

FL-1.4.3 (K2 Faire la différence entre les produits d'activités qui contribuent au processus de test

FL-1.4.4 (K2) Expliquer les bénéfices apportés par le maintien de la traçabilité entre les bases de test et les produits d'activités du test

1.5 La psychologie des tests

FL-1.5.1 (K1) Identifier les facteurs psychologiques ayant une influence sur le succès des tests

FL-1.5.2 (K2) Expliquer la différence entre l'état d'esprit requis pour les activités de test et l'état d'esprit requis pour les activités de développement

1.1 Que sont les tests ?

Les systèmes logiciels font partie intégrante de la vie quotidienne, depuis les applications commerciales (p. ex., les services bancaires) jusqu'aux produits pour les consommateurs (p. ex., voitures). La plupart des gens ont eu une expérience avec un logiciel qui n'a pas fonctionné comme prévu. Un logiciel qui ne fonctionne pas correctement peut entraîner de nombreux problèmes, y compris la perte d'argent, de temps ou entacher la réputation de l'entreprise, et même entraîner des blessures qui peuvent être mortelles. Les tests logiciels sont un moyen d'évaluer la qualité du logiciel et de réduire le risque de défaillance de ce logiciel en cours de fonctionnement.

Une perception erronée courante du test consiste à considérer que cela se réduit à exécuter des tests, c'est-à-dire exécuter le logiciel et en vérifier les résultats. Tel que décrit à la section 1.4, le test logiciel constitue un processus qui comprend de nombreuses activités. L'exécution des tests (y compris la vérification des résultats) n'est qu'une de ces activités. Le processus de test comprend également des activités telles que la planification des tests, l'analyse, la conception et la mise en œuvre des tests, le suivi de la progression et des résultats des tests, ainsi que l'évaluation de la qualité de l'objet de test.

Certains tests impliquent l'exécution du composant ou du système testé. Ces tests sont appelés tests dynamiques. D'autres tests n'impliquent pas l'exécution du composant ou du système testé ; de tels tests sont appelés tests statiques. Ainsi, les tests comprennent également la revue de produits d'activités tels que les exigences, les User Stories et le code source.

Une autre perception erronée courante sur le test est qu'il se concentre entièrement sur la vérification des exigences, des User Stories ou d'autres spécifications. Ainsi, si le test implique de vérifier que le système répond aux exigences spécifiées, il permet également de s'assurer que le système répondra aux besoins des utilisateurs et des autres parties prenantes dans son (ses) environnement(s) opérationnel(s).

Les activités de test sont organisées et réalisées différemment selon les différents cycles de vie (voir section 2.1).

1.1.1 Objectifs habituels des tests

Pour un projet donné, les objectifs du test peuvent inclure :

- Évaluer les produits d'activités tels que les exigences, les User Stories, la conception et le code
- Vérifier si toutes les exigences spécifiées ont été satisfaites
- Valider si l'objet de test est complet et fonctionne comme attendu par les utilisateurs et autres parties prenantes
- Construire la confiance dans le niveau de qualité de l'objet de test
- Prévenir des défauts
- Trouver des défaillances et défauts
- Fournir suffisamment d'information aux parties prenantes pour leur permettre de prendre des décisions éclairées, en particulier en ce qui concerne le niveau de qualité de l'objet de test
- Réduire le niveau de risque d'une qualité logicielle inadéquate (p. ex. des défaillances non détectées auparavant se produisant en opération)

- Se conformer aux exigences ou aux normes contractuelles, légales ou réglementaires, et/ou vérifier la compatibilité de l'objet de test avec de telles exigences ou normes

  Les objectifs de test peuvent varier en fonction du contexte du composant ou du système testé, du niveau de test et du modèle de cycle de vie de développement logiciel. Ces différences peuvent inclure, par exemple :

- Au cours des tests de composants, l'un des objectifs peut être de trouver autant de défaillances que possible de sorte que les défauts sous-jacents soient identifiés et corrigés tôt. Un autre objectif peut être d'augmenter la couverture du code des composants testés.
- Au cours des tests d'acceptation, l'un des objectifs peut être de confirmer que le système fonctionne comme prévu et satisfait aux exigences. Un autre objectif de ces tests peut être de fournir des informations aux parties prenantes sur le risque lié à la sortie d'une version du système à un moment donné.

1.1.2 Test et débogage

Les activités de tests et de débogage sont différentes. L'exécution de tests peut mettre en évidence des défaillances causées par des défauts dans le logiciel. Le débogage est l'activité de développement qui trouve, analyse et corrige de tels défauts. Par la suite, le test de confirmation vérifie si les corrections apportées ont résolu les défauts. Dans certains cas, les testeurs sont responsables du test initial et du test de confirmation final, tandis que les développeurs se chargent du débogage et du test des composants concernés. Cependant, dans le développement Agile et dans d'autres cycles de vie, les testeurs peuvent être impliqués dans le débogage et le test des composants.

La norme ISO (ISO/IEC/IEEE 29119-1) contient de plus amples informations sur les concepts des tests logiciels.

1.2 Pourquoi les tests sont-ils nécessaires ?

Des tests rigoureux des composants et des systèmes, ainsi que de leur documentation associée, peuvent aider à réduire le risque de défaillances pendant l'exploitation. Lorsque des défauts sont détectés et corrigés par la suite, cela contribue à la qualité des composants ou des systèmes. En outre, les tests logiciels peuvent également être nécessaires pour répondre à des exigences contractuelles ou légales ou à des normes spécifiques de l'industrie.

1.2.1 Contribution des tests au succès

Tout au long de l'histoire de l'informatique, il est assez courant que des logiciels et des systèmes soient mis en production et que, en raison de la présence de défauts, ils causent ensuite des défaillances ou ne satisfassent pas les besoins des parties prenantes. Pourtant, l'utilisation de techniques de test appropriées peut réduire la fréquence de ces livraisons problématiques, lorsque ces techniques sont appliquées avec le niveau approprié d'expertise en matière de tests, aux niveaux de test appropriés et aux moments appropriés du cycle de vie de développement logiciel. En voici quelques exemples :

- La participation des testeurs à la revue des exigences ou au raffinement des User Stories pourrait permettre de détecter des défauts dans ces produits d'activités. L'identification et l'élimination des défauts dans les exigences réduit le risque de développement de fonctionnalités incorrectes ou non testables.
- Le fait que les testeurs travaillent en étroite collaboration avec les concepteurs du système pendant la conception du système peut aider chaque partie à mieux comprendre la conception et

la façon de la tester. Cette meilleure compréhension peut réduire le risque de défauts de fond dans la conception et permettre d'identifier les tests à un stade précoce.

- Le fait que les testeurs travaillent en étroite collaboration avec les développeurs pendant que le code est en cours de développement peut augmenter la compréhension du code et de la façon de le tester par chaque partie. Cette meilleure compréhension peut réduire le risque de défauts dans le code et les tests.
- Le fait que les testeurs vérifient et valident le logiciel avant sa sortie permet de détecter des défaillances qui auraient pu être manquées et aide au processus d'élimination des défauts ayant causé les défaillances (c.-à-d. le débogage). Cela augmente la probabilité que le logiciel réponde aux besoins des parties prenantes et satisfasse les exigences.

En plus de ces exemples, l'atteinte des objectifs de test définis (voir la section 1.1.1) contribue au succès global du développement logiciel et de la maintenance.

1.2.2 Assurance qualité et test

Bien que les gens utilisent souvent l'expression assurance qualité (ou simplement QA – Quality Assurance) pour désigner le test, l'assurance qualité et le test ne sont pas les mêmes, mais ils sont liés. Un concept plus large, la gestion de la qualité, les lie l'un à l'autre. La gestion de la qualité comprend toutes les activités qui dirigent et contrôlent une organisation en matière de qualité. Entre autres activités, la gestion de la qualité comprend à la fois l'assurance de la qualité et le contrôle de la qualité. L'assurance qualité est principalement axée sur le respect des processus adéquats, afin de donner l'assurance que les niveaux de qualité appropriés seront atteints. Lorsque les processus sont correctement exécutés, les produits d'activités créés par ces processus sont généralement de meilleure qualité, ce qui contribue à la prévention des défauts. De plus, l'analyse des causes racines pour détecter et éliminer les causes des défauts, ainsi que l'application des résultats des réunions de rétrospective pour améliorer les processus, sont importantes pour une mise en œuvre efficace de l'assurance qualité.

Le contrôle de la qualité comprend diverses activités, y compris des activités de test, qui contribuent à l'atteinte de niveaux de qualité adéquats. Les activités de test font partie du processus global de développement ou de maintenance du logiciel. Puisque l'assurance qualité concerne l'exécution correcte de l'ensemble du processus, l'assurance de la qualité est un support à des tests pertinents. Comme décrit dans les sections 1.1.1 et 1.2.1, les tests contribuent à l'atteinte de la qualité de diverses façons.

1.2.3 Erreurs, défauts et défaillances

Une personne peut faire une erreur, ce qui peut conduire à l'introduction d'un défaut (faute ou bogue) dans le code du logiciel ou dans un autre produit d'activités connexe. Une erreur qui conduit à l'introduction d'un défaut dans un produit d'activités peut déclencher une erreur qui conduit à l'introduction d'un défaut dans un produit d'activités connexe. Par exemple, une erreur d'élucidation d'exigences peut conduire à un défaut au niveau des exigences, qui se traduit ensuite par une erreur de programmation qui conduit à un défaut dans le code.

Si un défaut dans le code est exécuté, cela peut causer une défaillance, mais pas nécessairement dans toutes les circonstances. Par exemple, certains défauts nécessitent des valeurs de données d'entrée ou des conditions préalables très spécifiques pour déclencher une défaillance, ce qui peut se produire rarement ou jamais.

Les erreurs peuvent survenir pour de nombreuses raisons, telles que :

- Les contraintes de temps
- La faillibilité humaine

- L'inexpérience ou le manque de compétence des participants au projet
- Une mauvaise communication entre les participants au projet, y compris au sujet des exigences et de la conception
- La complexité du code, de la conception, de l'architecture, du problème sous-jacent à résoudre, et/ou des technologies utilisées
- Les malentendus sur les interfaces intra-système et inter-système, en particulier lorsque de telles interfaces intra-système et inter-système sont très nombreuses
- Des technologies nouvelles, peu connues

En plus des défaillances causées par des défauts dans le code, les défaillances peuvent également être causées par les conditions environnementales. Par exemple, le rayonnement, les champs électromagnétiques et la pollution peuvent causer des défauts dans le firmware (logiciel système à bas niveau) ou influencer l'exécution du logiciel en changeant les conditions matérielles.

Les résultats inattendus ne sont pas tous des défaillances. Des faux positifs peuvent se produire en raison d'erreurs dans la façon dont les tests ont été exécutés, ou en raison de défauts dans les données de test, l'environnement de test, ou d'autres testware, ou pour d'autres raisons. La situation inverse peut également se produire, lorsque des erreurs ou des défauts similaires conduisent à de faux négatifs. Les faux négatifs sont des tests qui ne détectent pas les défauts qu'ils auraient dû détecter ; les faux positifs sont signalés comme des défauts, mais ne sont en réalité pas des défauts.

1.2.4 Défauts, causes racines et effets

Les causes racines des défauts sont les premières actions ou conditions qui ont contribué à la création des défauts. Les défauts peuvent être analysés pour identifier leurs causes racines, afin de réduire l'apparition de défauts similaires à l'avenir. En se concentrant sur les causes racines les plus importantes, l'analyse des causes racines peut conduire à des améliorations de processus qui préviennent l'introduction d'un nombre important de défauts futurs.

Par exemple, supposons que des paiements d'intérêts incorrects, dus à une seule ligne de code incorrecte, se traduisent par les plaintes des clients. Le code défectueux a été écrit pour une User Story qui était ambiguë, en raison de la mauvaise compréhension par le Product Owner de la façon de calculer les intérêts. S'il existe un pourcentage élevé de défauts dans les calculs d'intérêt, et que ces défauts ont leur cause racine dans des incompréhensions similaires, le Product Owner pourrait se former au calcul des intérêts afin de réduire le nombre de tels défauts à l'avenir.

Dans cet exemple, les plaintes des clients sont des effets. Les paiements d'intérêts incorrects sont des défaillances. Le calcul incorrect dans le code est un défaut, et il résulte du défaut d'origine, l'ambiguïté dans la User Story. La cause racine du défaut initial était un manque de connaissance de la part du Product Owner, ce qui l'a conduit à faire une erreur lors de la rédaction de la User Story. Le processus d'analyse des causes racines est traité dans les Syllabus de niveau expert ISTQB-ETM (Gestion des tests) et ISTQB-EITP (Améliorer le processus de test).

1.3 7 principes sur les tests

Un certain nombre de principes sur les tests ont été suggérés au cours des 50 dernières années et offrent des indications communes à tous les tests.

1. Les tests montrent la présence de défauts, par leur absence

Les tests peuvent prouver la présence de défauts, mais ne peuvent pas en prouver l'absence. Les tests réduisent la probabilité que des défauts restent cachés dans le logiciel mais, même si aucun défaut n'est découvert, ce n'est pas une preuve que tout est correct.

2. Les tests exhaustifs sont impossibles

Tout tester (toutes les combinaisons d'entrées et de préconditions) n'est pas faisable sauf pour des cas triviaux. Plutôt que de chercher à faire tests exhaustifs, l'analyse des risques, des techniques de test et des priorités devraient être utilisés pour cibler les efforts de tests.

3. Tester tôt économise du temps et de l'argent

Pour détecter tôt les défauts, à la fois des activités de tests statiques et des activités de tests dynamiques doivent être lancées le plus tôt possible dans le cycle de vie de développement du logiciel. Le fait de tester tôt est parfois appelé " shift left ". Tester tôt dans le cycle de vie du développement logiciel permet de réduire ou d'éliminer des changements coûteux (voir section 3.1).

4. Regroupement des défauts

Un petit nombre de modules contient généralement la plupart des défauts découverts lors des tests avant livraison, ou est responsable de la plupart des défaillances en exploitation. Des regroupements prévisibles de défauts, ou des regroupements réellement observés en test ou en exploitation, constituent un élément important de l'analyse des risques utilisée pour cibler l'effort de test (comme mentionné dans le principe 2).

5. Paradoxe du pesticide

Si les mêmes tests sont répétés de nombreuses fois, le même ensemble de cas de tests finira par ne plus détecter de nouveaux défauts. Pour détecter de nouveaux défauts, il peut être nécessaire de modifier les tests existants et les données de test existantes, ainsi que de rédiger de nouveaux tests. (Les tests ne sont plus efficaces pour détecter des défauts, tout comme les pesticides ne sont plus efficaces pour tuer les insectes après un certain temps). Dans certains cas, comme les tests de régression automatisés, le paradoxe du pesticide a un résultat bénéfique, qui est le nombre relativement faible de défauts de régression.

6. Les tests dépendent du contexte

Les tests sont effectués différemment dans des contextes différents. Par exemple, un logiciel de contrôle industriel, critique au niveau de la sureté, sera testé différemment d'une application de commerce électronique sur téléphone mobile. Comme autre exemple, le test dans un projet Agile est effectué différemment du test dans un projet à cycle de vie séquentiel (voir section 2.1).

7. L'absence d'erreurs est une illusion

Certaines organisations s'attendent à ce que les testeurs puissent effectuer tous les tests possibles et trouver tous les défauts possibles, mais les principes 2 et 1, respectivement, nous disent que c'est impossible. De plus, il est illusoire (c.-à-d. une croyance erronée) de s'attendre à ce que le simple fait de trouver et de corriger un grand nombre de défauts garantisse la réussite d'un système. Par exemple, tester en profondeur toutes les exigences spécifiées et corriger tous les défauts trouvés pourrait toujours produire un système qui est difficile à utiliser, qui ne répond pas aux besoins et aux attentes des utilisateurs ou qui est moins performants comparé à d'autres systèmes concurrents.

Voir Myers 2011, Kaner 2002 et Weinberg 2008 pour des exemples de ces principes de test et d'autres principes de test.

1.4 Processus de test

Il n'existe pas un unique processus de test logiciel universel, mais il existe des ensembles communs d'activités de test sans lesquels les tests auront moins de chances d'atteindre les objectifs fixés. Ces ensembles d'activités de test constituent un processus de test. Le processus de test logiciel approprié et spécifique dans une situation donnée dépend de nombreux facteurs. Quelles activités de test sont impliquées dans ce processus, comment ces activités sont mises en œuvre et quand ces activités doivent être réalisées, peut être précisé dans la stratégie de test d'une organisation.

1.4.1 Le processus de test dans le contexte

Les facteurs contextuels qui influencent le processus de test d'une organisation comprennent, sans toutefois s'y limiter :

- Le modèle de cycle de vie du développement logiciel et les méthodologies de projet utilisées
- Les niveaux de test et types de test prévus
- Les risques liés aux produits et aux projets
- Le domaine d'activité
- Les contraintes opérationnelles, entre autres :
  - Les budgets et ressources
  - Les délais
  - La complexité
  - Les exigences contractuelles et réglementaires
- Les politiques et pratiques organisationnelles
- Les normes internes et externes requises

Les sections suivantes décrivent les aspects généraux des processus de test selon les aspects suivants :

- Activités et tâches de test
- Produits d'activités du test
- Traçabilité entre les bases de test et les produits d'activités du test

Il est très utile que les bases de test (pour n'importe quel niveau ou type de test considéré) aient des critères de couverture mesurables définis. Les critères de couverture peuvent servir efficacement d'indicateurs clés de performance (KPI) pour guider les activités qui démontrent l'atteinte des objectifs des tests logiciels (voir section 1.1.1).

Par exemple, pour une application mobile, les bases de test peuvent inclure une liste d'exigences et une liste d'appareils mobiles pris en charge. Chaque exigence est un élément de la base de test. Chaque appareil supporté est également un élément des bases de test. Les critères de couverture peuvent exiger au moins un cas de test pour chaque élément des bases de test. Une fois exécutés, les résultats de ces tests indiquent aux parties prenantes si les exigences spécifiées sont satisfaites et si des défaillances ont été observées sur les appareils pris en charge.

La norme ISO (ISO/IEC/IEEE 29119-2) contient des informations complémentaires sur les processus de test.

1.4.2 Activités et taches de test

Un processus de test se compose des principaux groupes d'activités suivants :

- Planification des tests
- Suivi et contrôle des tests
- Analyse de test
- Conception des tests
- Implémentation des tests
- Exécution des tests
- Clôture des tests

Chaque groupe d'activités est composé d'activités constitutives, qui seront décrites dans les sous-sections ci-dessous. Chaque activité au sein de chaque groupe d'activités peut à son tour consister en de multiples tâches individuelles, qui peuvent varier d'un projet ou d'une version à une autre.

De plus, même si plusieurs de ces groupes d'activités peuvent sembler logiquement séquentiels, ils sont souvent mis en œuvre de façon itérative. Par exemple, le développement Agile implique de petites itérations de conception de logiciels, de construction et de test qui se produisent en continu, soutenues par une planification régulière. Ainsi, les activités de test se déroulent également de façon itérative et continue dans le cadre de cette approche de développement. Même en mode séquentiel, la séquence logique par étapes des activités induira le chevauchement, la combinaison, la concomitance, ou l'omission, de sorte qu'une adaptation de ces activités principales dans le contexte du système et du projet est habituellement nécessaire.

Planification des tests

La planification des tests implique de définir les objectifs du test et l'approche retenue pour atteindre les objectifs du test dans le respect des contraintes imposées par le contexte (p. ex. spécifier les techniques et tâches de test appropriées, et produire un calendrier de test pour respecter une date limite). Les plans de test peuvent être revus en fonction des retours sur les activités de pilotage et de contrôle. La planification des tests est expliquée plus en détail à la section 5.2.

Pilotage et contrôle des tests

Le pilotage des tests implique la comparaison régulière de l'avancement réel par rapport au plan de test à l'aide des métriques de pilotage définies dans le plan de test. Le contrôle des tests consiste à prendre les mesures nécessaires pour satisfaire aux objectifs du plan de test (qui peut être mis à jour au fil du temps). Le pilotage et le contrôle des tests se basent sur l'évaluation des critères de sortie, que l'on appelle « definition of done » (définition de « terminé ») dans certains cycles de vie (cf. syllabus Testeur Certifié - Extension Niveau Fondation Testeur Agile CFTL / ISTQB). Par exemple, l'évaluation des critères de sortie pour l'exécution des tests dans le cadre d'un niveau de test donné peut inclure :

- La vérification des résultats et logs des tests de test par rapport aux critères de couverture spécifiés
- L'évaluation du niveau de qualité des composants ou du système sur la base des résultats et logs des tests
- La détermination de la nécessité d'autres tests (p. ex. si les tests qui visaient à l'atteinte d'un certain niveau de couverture des risques Produit n'y sont pas parvenu, il est nécessaire que des tests supplémentaires soient écrits et exécutés)

L'avancement des tests par rapport au plan est communiqué aux parties prenantes dans des rapports d'avancement des tests, comprenant les écarts par rapport au plan et les informations utiles à toute décision d'arrêter les tests.

Le pilotage et le contrôle des tests sont expliqués plus en détail à la section 5.3.

Analyse de test

Pendant l'analyse de test, les bases de test sont analysées pour identifier les caractéristiques testables et définir les conditions de test associées. En d'autres termes, l'analyse des tests détermine "quoi tester" en termes de critères de couverture mesurables.

L'analyse de test comprend les principales activités suivantes :

- Analyser les bases de test appropriées au niveau de test considéré, par exemple :
  - Les spécifications des exigences, telles que les exigences métier, les exigences fonctionnelles, les exigences système, les User Stories, les epics, les cas d'utilisation ou les produits d'activités similaires qui spécifient le comportement fonctionnel et non-fonctionnel souhaité pour le composant ou système
  - Les informations sur la conception et l'implémentation, comme les diagrammes ou documents d'architecture du système ou du logiciel, les spécifications de conception, les flux d'appels, les diagrammes de modélisation (p. ex., les diagrammes UML ou entité-relation), les spécifications d'interface ou des produits d'activités similaires qui spécifient la structure du composant ou du système
  - L'implémentation du composant ou du système lui-même, y compris le code, les métadonnées et les requêtes sur la base de données, et les interfaces
  - Les rapports d'analyse des risques, qui peuvent prendre en compte les aspects fonctionnels, non-fonctionnels et structurels du composant ou du système
- Evaluer les bases de test et des éléments de test pour identifier des défauts de différents types, tels que :
  - Ambiguïtés
  - Omissions
  - Incohérences
  - Inexactitudes
  - Contradictions
  - Déclarations superflues
- Identifier les caractéristiques et les ensembles de caractéristiques à tester
- Définir et prioriser les conditions de test pour chaque caractéristique en fonction de l'analyse des bases de test et en tenant compte des caractéristiques fonctionnelles, non-fonctionnelles et structurelles, des autres facteurs métier et techniques, et des niveaux de risque
- Capturer la traçabilité bidirectionnelle entre chaque élément des bases de test et les conditions de test associées (voir sections 1.4.3 et 1.4.4)

L'application de techniques de test boîte-noire, boîte-blanche et basées sur l'expérience peut s'avérer utile dans le processus d'analyse de test (voir chapitre 4) pour réduire la probabilité d'omettre des conditions de test importantes, et pour définir des conditions de test plus exactes et plus précises.

Dans certains cas, l'analyse de test produit des conditions de test qui doivent être utilisées comme objectifs de test dans des chartes de test. Les chartes de test sont des produits d'activités typiques dans certains types de tests basés sur l'expérience (voir la section 4.4.2). Lorsque ces objectifs de test sont traçables avec les bases de test, la couverture atteinte au cours de ce type de tests basés sur l'expérience peut être mesurée.

L'identification de défauts au cours de l'analyse de test est un bénéfice potentiel important, en particulier lorsqu'il n'y a pas d'autre processus de revue utilisé et/ou si le processus de test est étroitement lié au processus de revue. De telles activités d'analyse de test ne se contentent pas de vérifier si les exigences sont cohérentes, correctement exprimées, et complètes, mais aussi de valider si les exigences capturent correctement les besoins des clients, utilisateurs et des autres parties prenantes. Par exemple, les techniques telles que le BDD (Behavior-Driven Developement) et ATDD (Acceptance Test-Driven Development), qui impliquent la définition des conditions de test et de cas de test à partir des User Story et des critères d'acceptation avant le codage, permettent aussi de vérifier, valider et détecter des défauts dans les User Stories et les critères d'acceptation (cf. syllabus Testeur Certifié - Extension Niveau Fondation Testeur Agile CFTL / ISTQB).

Conception des tests

Lors de la conception des tests, les conditions de test sont déclinées en cas de test de haut niveau, en ensembles de cas de tests de haut niveau et autres testware. Ainsi, l'analyse de test répond à la question « quoi tester ? » alors que la conception des tests répond à la question « comment tester ? ».

La conception des tests comprend les activités principales suivantes :

- Concevoir et prioriser les cas de test et les ensembles de cas de test
- Identifier les données de test nécessaires pour les conditions de test et les cas de test
- Concevoir l'environnement de test et identifier l'infrastructure et les outils nécessaires
- Etablir la traçabilité bidirectionnelle entre les bases de test, les conditions de test, les cas de test et les procédures de test (voir section 1.4.4)

La déclinaison des conditions de test en cas de test et ensembles de cas de test lors de la conception des tests implique souvent l'utilisation de techniques de test (voir chapitre 4).

Comme pour l'analyse de test, la conception des tests peut également aboutir à l'identification de types similaires de défauts dans les bases de test. De même, comme pour l'analyse de test, l'identification de défauts lors de la conception des tests est un potentiel important de bénéfice.

Implémentation des tests

Au cours de l'implémentation des tests, le testware nécessaire à l'exécution des tests est créé et/ou complété, y compris l'ordonnancement des cas de test en procédures de test. Ainsi, la conception des tests répond à la question « comment tester ? » tandis que l'implémentation des tests répond à la question « est-ce que tout est en place pour exécuter les tests ? ».

L'implémentation des tests comprend les activités principales suivantes :

- Développer et prioriser les procédures de test, et, éventuellement, créer des scripts de test automatisés
- Créer des suites de tests à partir des procédures de test et (le cas échéant) des scripts de tests automatisés
- Positionner les suites de tests dans un calendrier d'exécution des tests de manière à obtenir une exécution efficace des tests (voir section 5.2.4)

- Construire l'environnement de test (y compris, potentiellement, les harnais de test, la virtualisation des services, les simulateurs et d'autres éléments d'infrastructure) et vérifier que tout le nécessaire a été correctement mis en place
- Préparer les données de test et s'assurer qu'elles sont correctement chargées dans l'environnement de test
- Vérifier et mettre à jour la traçabilité bidirectionnelle entre les bases de test, les conditions de test, les cas de test, les procédures de test et les suites de tests (voir section 1.4.4)

Les tâches de conception et d'implémentation des tests sont souvent combinées.

Dans le cas des tests exploratoires et d'autres types de tests basés sur l'expérience, la conception et l'implémentation des tests peuvent être réalisées et éventuellement être documentées lors de l'exécution des tests. Les tests exploratoires peuvent être basés sur des chartes de test (produites dans le cadre de l'analyse de test), et les tests exploratoires sont exécutés immédiatement, en même temps que leur conception et leur implémentation (voir section 4.4.2).

Exécution des tests

Pendant l'exécution des tests, les suites de tests sont exécutées conformément au calendrier d'exécution des tests.

L'exécution des tests comprend les activités principales suivantes :

- Enregistrer les IDs et les versions des éléments de test ou de l'objet de test, des outils de test et des testware
- Exécuter les tests manuellement ou à l'aide d'outils d'exécution de test
- Comparer les résultats obtenus avec les résultats attendus
- Analyser les anomalies afin d'établir leurs causes probables (p. ex., des défaillances peuvent survenir en raison de défauts dans le code, mais de faux positifs peuvent également se produire [voir section 1.2.3])
- Signaler les défauts sur la base des défaillances observées (voir section 5.6)
- Enregistrer les résultats de l'exécution des tests (par exemple, réussite, échec, blocage)
- Répéter certaines activités de test, soit à la suite d'une action prise pour une anomalie, soit dans le cadre de l'activité planifiée de test (p. ex. exécution d'un test corrigé, test de confirmation, et/ou test de régression)
- Vérifier et mettre à jour la traçabilité bidirectionnelle entre les bases de test, les conditions de test, les cas de test, les procédures de test et les résultats des tests

Clôture des tests

Les activités de clôture des tests collectent les données des activités de test terminées afin de consolider l'expérience, les testware et toute autre information pertinente. Les activités de clôture des tests ont lieu à des jalons du projet, par exemple lorsqu'un système logiciel est livré, qu'un projet de test est terminé (ou annulé), qu'une itération de projet Agile est terminée (par exemple, dans le cadre d'une réunion rétrospective), qu'un niveau de test est terminé ou qu'une maintenance de version est terminée.

La clôture des tests comprend les activités principales suivantes :

- Vérifier si tous les rapports de défauts sont clôturés, et saisir des demandes de modification ou des items du backlog du produit pour tous les défauts non résolus à la fin de l'exécution des tests

- Créer un rapport de synthèse de test à communiquer aux parties prenantes
- Finaliser et archiver l'environnement de test, les données de test, l'infrastructure de test et autres testware pour une réutilisation ultérieure
- Remettre le testware aux équipes de maintenance, aux autres équipes de projet, et/ou à d'autres parties prenantes qui pourraient bénéficier de son utilisation
- Analyser les leçons apprises des activités de test terminées afin de déterminer les changements nécessaires pour les itérations, versions et projets futurs
- Utiliser l'information recueillie pour améliorer la maturité du processus de test

1.4.3 Les produits d'activités du test

Les produits d'activités du test sont créés dans le cadre du processus de test. Tout comme il y a d'importantes variations dans la façon dont les organisations mettent en œuvre le processus de test, il y a aussi des variations importantes dans les types de produits d'activités créés au cours de ce processus, dans la façon dont ces produits d'activités sont organisés et gérés, et dans les noms utilisés pour ces produits d'activités. Ce syllabus s'appuie sur le processus de test décrit précédemment, et sur les produits d'activités décrits dans ce syllabus et dans le glossaire de l'ISTQB. La norme ISO (ISO/IEC/IEEE 29119-3) peut également servir de référence pour les produits d'activités du test.

De nombreux produits d'activités du test décrit dans cette section peuvent être saisis et gérés à l'aide d'outils de gestion des tests et d'outils de gestion des défauts (voir chapitre 6).

Produits d'activités de la planification des tests

Les produits d'activités de la planification des tests comprennent généralement un ou plusieurs plans de test. Le plan de test comprend des informations sur les bases de test, auxquelles les autres produits des activités du test seront liés via les informations de traçabilité (voir ci-dessous et section 1.4.4), ainsi que les critères de sortie ou définition du terminé (« definition of done » en anglais) qui seront utilisés lors du pilotage et du contrôle des tests. Les plans de test sont décrits dans la section 5.2.

Produits d'activités du pilotage et contrôle des tests

Les produits d'activités du pilotage et contrôle des tests comprennent habituellement divers types de rapports de test, y compris les rapports d'avancement des tests (produits sur une base continue et/ou régulière) et les rapports de synthèse des tests (produits de façon continue et/ou régulière). Tous les rapports de test devraient fournir des détails pertinents pour les parties prenantes ciblées au sujet de l'avancement des tests à la date du rapport, y compris un résumé des résultats de l'exécution des tests lorsque ceux-ci sont disponibles.

Les produits d'activités du pilotage et contrôle des tests devraient également tenir compte des aspects spécifiques relatifs à la gestion de projet, comme l'achèvement des tâches, l'affectation et l'utilisation des ressources, et l'effort.

Le pilotage et le contrôle des tests, ainsi que les produits d'activités créés durant ces activités, sont expliqués plus en détail à la section 5.3 de ce syllabus.

Produits d'activités de l'analyse de test

Les produits d'activités de l'analyse de test comprennent les conditions de test définies et priorisées, dont chacune est idéalement traçable de façon bidirectionnelle avec le ou les élément(s) spécifiques des bases de test qu'elle couvre. Pour les tests exploratoires, l'analyse de test peut impliquer la création de chartes de test. L'analyse de test peut également aboutir à la découverte de défauts dans les bases de test et à leur la documentation.

Produits d'activités de la conception des tests

La conception des tests a pour résultats des cas de test et des ensembles de cas de test pour exercer les conditions de test définies dans l'analyse de test. La conception de cas de tests de haut niveau, sans valeurs concrètes pour les données d'entrée et les résultats attendus, constitue souvent une bonne pratique. De tels cas de test de haut niveau sont réutilisables sur plusieurs cycles de test avec des données concrètes différentes, tout en documentant de manière adéquate la portée du cas de test. Idéalement, chaque cas de test est traçable de façon bidirectionnelle avec la ou les condition(s) de test qu'il couvre.

La conception des tests aboutit également à la conception et/ou à l'identification des données de test nécessaires, à la conception de l'environnement de test et à l'identification de l'infrastructure et des outils, bien que la façon avec laquelle ces résultats sont documentés puisse varier amplement.

Les conditions de test définies dans l'analyse des tests peuvent être ultérieurement affinées lors de la conception des tests.

Produits d'activités de l'implémentation des tests

Les produits d'activités de l'implémentation des tests incluent :

- Les procédures de test et l'ordonnancement de ces procédures de test
- Les suites de test
- Un calendrier d'exécution des tests

Idéalement, une fois l'implémentation des tests terminée, l'atteinte des critères de couverture établis dans le plan de test peut être démontrée par une traçabilité bidirectionnelle entre les procédures de test et des éléments spécifiques des bases de test, au travers des cas de test et des conditions de test.

Dans certains cas, l'implémentation des tests implique la création de produits d'activités utilisant ou utilisés par des outils, tels que la virtualisation de services et les scripts de test automatisés.

L'implémentation des tests peut également aboutir à la création et à la vérification de données de test et de l'environnement de test. L'exhaustivité de la documentation des résultats de la vérification des données et/ou de l'environnement peut varier considérablement.

Les données de test sont utilisées pour attribuer des valeurs concrètes aux entrées et aux résultats attendus des cas de test. Ces valeurs concrètes, accompagnées d'instructions explicites sur la façon de les utiliser, transforment des cas de test de haut niveau en cas de test de bas niveau exécutables. Le même cas de test de haut niveau peut utiliser des données de test différentes lorsqu'il est exécuté sur des versions différentes de l'objet de test. Les résultats attendus concrets qui sont associés à des données concrètes de test sont identifiés à l'aide d'un oracle de test.

Dans les tests exploratoires, certains produits d'activités de la conception et de l'implémentation des tests peuvent être créés pendant l'exécution des tests, bien que la mesure dans laquelle les tests exploratoires (et leur traçabilité à des éléments spécifiques des bases de test) sont documentés peut varier considérablement.

Les conditions de test définies dans l'analyse de test peuvent être affinées lors de l'implémentation des tests.

Produits d'activités de l'exécution des tests

Les produits d'activités de l'exécution des tests comprennent :

- La documentation du statut de chaque cas de test ou des procédures de test (p. ex. prêt à être exécuté, réussite, échec, blocage, omission délibérée, etc.)

- Les rapports de défauts (voir section 5.6)
- La documentation précisant quel(s) élément(s) de test, objet(s) de test, outils de test et testware ont été impliqués dans le test

Idéalement, une fois l'exécution des tests terminée, le statut de chaque élément des bases de test peut être déterminé et indiqué par traçabilité bidirectionnelle avec la ou les procédures de test associée(s). Par exemple, nous pouvons dire quelles exigences ont réussi tous les tests planifiés, quelles exigences ont échoué aux tests et/ou ont des défauts qui leur sont associés et quelles exigences ont des tests planifiés encore en attente d'être exécutés. Cela permet de vérifier que les critères de couverture ont été satisfaits et de rendre compte des résultats des tests de façon compréhensible par les parties prenantes.

Produits d'activités de la clôture des tests

Les produits d'activités de la clôture des tests comprennent les rapports de synthèse de test, les mesures à prendre pour améliorer les projets ou les itérations ultérieurs (p. ex. à la suite d'une rétrospective en projet Agile), des demandes de changement ou des items du backlog du produit, et des testware finalisés.

1.4.4 Traçabilité entre les bases de test et les produits d'activités du test

Tel que mentionné à la section 1.4.3, les produits d'activités du test et les noms de ces produits varient considérablement. Indépendamment de ces variations, afin de mettre en œuvre un pilotage et un contrôle efficaces des tests, il est important d'établir et de maintenir la traçabilité tout au long du processus de test entre chaque élément des bases de test et les divers produits d'activités du test associés à cet élément, comme décrit ci-dessus. En plus de l'évaluation de la couverture de test, une bonne traçabilité facilite :

- L'analyse de l'impact des changements
- L'audit des tests
- La satisfaction des critères de gouvernance IT
- L'amélioration du caractère compréhensible des rapports d'avancement des tests et des rapports de synthèse de test afin d'y inclure l'état des éléments des bases de test (p. ex. les exigences qui ont été testées avec succès, les exigences qui ont des tests en échec, et les exigences qui ont des tests en attente)
- La restitution des aspects techniques des tests aux parties prenantes en des termes qu'elles peuvent comprendre
- L'apport d'information pour évaluer la qualité du produit, l'aptitude du processus et l'avancement du projet par rapport aux objectifs métier

Certains outils de gestion des tests fournissent des modèles de produits d'activités de test qui correspondent à une partie ou à la totalité des produits d'activités de test décrits dans cette section. Certaines organisations construisent leurs propres systèmes de gestion pour organiser les produits d'activités et assurer la traçabilité de l'information dont elles ont besoin.

1.5 La psychologie des tests

Le développement de logiciels, y compris les tests logiciels, implique des êtres humains. Par conséquent, la psychologie humaine a des effets importants sur les tests logiciels.

1.5.1 Psychologie humaine et test

Identifier les défauts lors de test statique tel qu'une revue des exigences ou une session de raffinement des User Stories, ou l'identification de défaillances au cours de l'exécution de tests dynamiques peut être perçu comme une critique du produit et de ses auteurs. Un élément de la psychologie humaine appelé biais de confirmation peut rendre difficile à accepter des informations en désaccord avec les croyances actuelles. Par exemple, puisque les développeurs attendent de leur code qu'il soit correct, ils ont un biais de confirmation qui fait qu'il est difficile d'accepter que le code est incorrect. Outre le biais de confirmation, d'autres biais cognitifs peuvent rendre difficile à comprendre ou à accepter l'information produite par les tests. De plus, c'est un trait humain commun de blâmer le porteur de la mauvaise nouvelle, et l'information produite par le test contient souvent de mauvaises nouvelles.

En raison de ces facteurs psychologiques, certaines personnes peuvent percevoir le test comme une activité destructrice, même s'il contribue grandement à l'avancement du projet et à la qualité du produit (voir les sections 1.1 et 1.2). Pour tenter de réduire ces perceptions, l'information sur les défauts et les défaillances devrait être communiquée de façon constructive. De cette façon, les tensions entre les testeurs et les analystes, les Product Owners, les concepteurs et les développeurs peuvent être réduites. Ceci s'applique aussi bien lors des tests statiques que dynamiques.

Les testeurs et les Test Managers doivent avoir de bonnes compétences interpersonnelles pour être capable de communiquer efficacement sur les défauts, les défaillances, les résultats des tests, l'avancement des tests et les risques, et pour établir des relations positives avec leurs collègues. Les moyens de bien communiquer comprennent les exemples suivants :

- Commencer par la collaboration plutôt que par la confrontation. Rappeler à tous l'objectif commun d'une meilleure qualité des systèmes.
- Mettre l'accent sur les bénéfices du test. Par exemple, pour les auteurs, l'information sur les défauts peut les aider à améliorer leurs produits d'activités et leurs compétences. Pour l'organisation, les défauts trouvés et corrigés durant les tests permettront d'économiser du temps et de l'argent et de réduire le risque global pour la qualité du produit.
- Communiquer les résultats des tests et d'autres constats d'une manière neutre et axée sur les faits, sans critiquer la personne qui a créé l'item défectueux. Rédiger des rapports objectifs et factuels sur les défauts et les constats des revues.
- Essayer de comprendre ce que ressent l'autre personne et les raisons pour lesquelles elle peut réagir négativement à l'information.
- Confirmer que l'autre personne a compris ce qui a été dit et vice versa.

Les objectifs habituels des tests ont été discutés plus tôt (voir section 1.1). Définir clairement le bon ensemble d'objectifs de test a d'importantes implications psychologiques. La plupart des gens ont tendance à aligner leurs plans et leurs comportements avec les objectifs fixés par l'équipe, la direction et les autres parties prenantes. Il est également important que les testeurs adhèrent à ces objectifs avec un biais personnel minimum.

1.5.2 Etat d'esprit des testeurs et des développeurs

Les développeurs et les testeurs pensent souvent différemment. L'objectif premier du développement est de concevoir et de construire un produit. Comme nous l'avons déjà mentionné, les objectifs du test comprennent la vérification et la validation du produit, la détection des défauts avant la livraison, et ainsi de suite. Il s'agit d'ensembles différents d'objectifs qui exigent des états d'esprit différents. L'union de ces points de vue permet d'atteindre un niveau plus élevé de qualité des produits.

Un état d'esprit reflète les hypothèses d'une personne et les méthodes qu'elle préfère pour la prise de décision et la résolution de problèmes. L'état d'esprit d'un testeur devrait inclure la curiosité, un pessimisme professionnel, un œil critique, l'attention aux détails et une motivation pour de bonnes et positives communications et relations. L'état d'esprit d'un testeur tend à grandir et à mûrir au fur et à mesure que le testeur acquiert de l'expérience.

L'état d'esprit d'un développeur peut inclure certains éléments de l'état d'esprit d'un testeur, mais les développeurs qui réussissent sont souvent plus intéressés par la conception et la construction de solutions que par la recherche de ce qui pourrait être erroné avec ces solutions. De plus, le biais de confirmation fait qu'il est difficile de trouver des erreurs dans son propre travail.

Avec le bon état d'esprit, les développeurs sont capables de tester leur propre code. Différents modèles de cycle de vie de développement du logiciel ont souvent des façons différentes d'organiser les testeurs et les activités de test. Le fait que certaines des activités de test soient effectuées par des testeurs indépendants augmente l'efficacité de la détection des défauts, ce qui est particulièrement important pour les systèmes de grande taille, complexes ou critiques sur le plan de la sureté. Les testeurs indépendants apportent une perspective différente de celle des auteurs du produit d'activités testé (c.-à-d. les analystes métier, les Product Owners, les concepteurs et les développeurs), puisqu'ils ont des biais cognitifs différents de ceux des auteurs.

2 Tester pendant le cycle de vie du développement logiciel — 100 minutes

Termes

test d'acceptation, test alpha, test bêta, logiciel commercial sur étagère (COTS), test d'intégration de composants, test de composants, test de confirmation, test d'acceptation contractuelle, test fonctionnel, analyse d'impact, test d'intégration, test de maintenance, test non-fonctionnel, test d'acceptation opérationnelle, test de régression, test d'acceptation réglementaire, modèle de développement séquentiel, test d'intégration de systèmes, test système, bases de test, cas de test, environnement de test, niveau de test, objet de test, objectif de test, type de test, test d'acceptation utilisateur, test boîte-blanche

Objectifs d'apprentissage pour Tester pendant le cycle de vie du développement logiciel

2.1 Les modèles de développement logiciel

FL-2.1.1 (K2) Expliquer les relations entre les activités de développement logiciel et les activités de test dans le cycle de vie du développement logiciel

FL-2.1.2 (K1Identifier les raisons qui font que les modèles de développement logiciel doivent être adaptés au contexte du projet et aux caractéristiques du produit

2.2 Niveaux de tests

FL-2.2.1 (K2) Comparer les différents niveaux de test en termes d'objectifs, de bases de test, d'objets de test, de défauts et de défaillances typiques, d'approches et de responsabilités

2.3 Types de tests

FL-2.3.1 (K2) Comparer les tests fonctionnels, non-fonctionnels et boîte-blanche

FL-2.3.2 (K1) Reconnaître que les tests fonctionnels, non-fonctionnels et boîte-blanche peuvent se produire à n'importe quel niveau de test

FL-2.3.3 (K2) Comparer les objectifs des tests de confirmation et des tests de régression

2.4 Tests de maintenance

FL-2.4.1 (K2) Résumer les événements déclencheurs des tests de maintenance

FL-2.4.2 (K2) Décrire le rôle de l'analyse d'impact dans les tests de maintenance

2.1 Les modèles de développement logiciel

Un modèle de cycle de vie de développement logiciel décrit les types d'activités réalisées à chaque étape d'un projet de développement logiciel, et la façon dont les activités sont reliées entre elles logiquement et chronologiquement. Il existe un certain nombre de modèles de cycle de développement de logiciels différents, chacun d'entre eux nécessitant des approches de test différentes.

2.1.1 Développement de logiciel et tests logiciels

Une partie importante du rôle d'un testeur est de se familiariser avec les principaux modèles de cycle de vie du développement logiciel afin que les activités de test adaptées puissent être réalisées.

Quel que soit le modèle de cycle de vie de développement logiciel, il y a plusieurs caractéristiques de bonnes pratiques des tests :

- Pour chaque activité de développement, il y a une activité de test correspondante
- Chaque niveau de test a des objectifs de test spécifiques à ce niveau
- L'analyse et la conception des tests pour un niveau de test donné commencent au cours de l'activité de développement correspondante
- Les testeurs participent aux discussions pour définir et affiner les exigences et la conception, et sont impliqués dans la revue des produits d'activités (p. ex. les exigences, la conception, les User Stories, etc.) dès que des versions préliminaires sont disponibles

Quel que soit le modèle de cycle de développement logiciel choisi, les activités de test devraient commencer dès les premières étapes du cycle de vie, en respectant le principe de « Tester tôt ».

Ce syllabus catégorise les modèles de cycle de vie de développement de logiciels courants comme suit :

- Modèles de développement séquentiel
- Modèles de développement itératif et incrémental

Un modèle de développement séquentiel décrit le processus de développement logiciel comme un flux linéaire et séquentiel d'activités. Cela signifie que toute phase du processus de développement devrait commencer lorsque la phase précédente est terminée. En théorie, il n'y a pas de chevauchement des phases, mais dans la pratique, il est bénéfique d'avoir un retour rapide de la phase suivante.

Dans le modèle 'en cascade', les activités de développement (p. ex. analyse des exigences, conception, codage, test) sont réalisées l'une après l'autre. Dans ce modèle, les activités de test ont seulement lieu une fois que toutes les autres activités de développement sont terminées.

Contrairement au modèle en 'cascade', le modèle en V intègre le processus de test tout au long du processus de développement, en appliquant le principe de « Tester tôt ». De plus, le modèle en V inclut des niveaux de test associés à chaque phase de développement correspondante, ce qui favorise le « Tester tôt » (voir la section 2.2 pour une discussion sur les niveaux de test). Dans ce modèle, l'exécution des tests associés à chaque niveau de test se déroule séquentiellement, mais dans certains cas, il y a chevauchement.

Les modèles de développement séquentiels fournissent des logiciels qui contiennent l'ensemble des fonctionnalités, mais qui nécessitent généralement des mois ou des années pour être livrés aux parties prenantes et aux utilisateurs.

Le développement incrémental implique la définition des exigences, la conception, le développement et le test d'un système par morceaux, ce qui signifie que les fonctionnalités du logiciel augmentent de façon

incrémentale. La taille de ces incréments de fonctionnalités varie, certaines méthodes ayant des étapes plus grandes et d'autres plus petites. Les incréments de caractéristiques peuvent être aussi petits qu'une unique modification d'un écran d'interface utilisateur ou qu'une nouvelle option de requête.

Le développement itératif se déroule lorsque des groupes de caractéristiques sont spécifiés, conçus, développés et testés ensemble dans une série de cycles, souvent d'une durée fixe. Les itérations peuvent impliquer des changements sur les caractéristiques développées dans les itérations précédentes, ainsi que des changements sur le périmètre du projet. Chaque itération délivre un logiciel opérationnel qui est un sous-ensemble croissant de l'ensemble global des caractéristiques jusqu'à ce que le logiciel final soit livré ou que le développement soit arrêté.

En voici quelques exemples :

- Rational Unified Process : chaque itération a tendance à être relativement longue (p. ex. deux à trois mois), et les incréments de caractéristiques sont proportionnellement importants, de l'ordre de deux ou trois groupes de caractéristiques connexes
- Scrum : chaque itération a tendance à être relativement courte (p. ex., heures, jours ou quelques semaines), et les incréments de caractéristiques sont proportionnellement petits, de l'ordre de quelques améliorations et/ou deux ou trois nouvelles caractéristiques
- Kanban : mis en œuvre avec des itérations de durée fixe ou non, pouvant soit livrer à la fin une seule amélioration ou caractéristique, soit regrouper des groupes de caractéristiques pour les livrer en une fois
- Spiral (ou par prototypage) : il s'agit de créer des incréments expérimentaux, dont certains peuvent être fortement remaniés ou même abandonnés lors de travaux de développement ultérieurs

Les composants ou systèmes développés à l'aide de ces méthodes impliquent souvent le chevauchement et l'itération sur les niveaux de test tout au long du développement. Idéalement, chaque caractéristique est testée à plusieurs niveaux de test au fur et à mesure qu'elle se rapproche de la livraison. Dans certains cas, les équipes utilisent la livraison continue ou le déploiement continu, les deux impliquant une automatisation significative de multiples niveaux de test dans le cadre de leurs flux de livraison. De nombreux efforts de développement utilisant ces méthodes incluent également le concept d'équipes auto-organisées, ce qui peut changer la façon dont le travail de test est organisé ainsi que la relation entre les testeurs et les développeurs.

Ces méthodes produisent un système aux caractéristiques croissantes, qui peut être livré aux utilisateurs finaux caractéristique par caractéristique, itération par itération, ou de façon plus traditionnelle par version majeure. Que les incréments logiciels soient livrés aux utilisateurs finaux ou non, les tests de régression sont de plus en plus importants à mesure que le système grossit.

Contrairement aux modèles séquentiels, les modèles itératifs et incrémentaux peuvent fournir des logiciels utilisables en quelques semaines ou même en quelques jours, mais ne peuvent livrer l'ensemble des exigences que sur une période de plusieurs mois ou même années.

Pour plus d'informations sur les tests logiciels dans le contexte du développement Agile, voir le syllabus Testeur Certifié - Extension Niveau Fondation Testeur Agile CFTL / ISTQB, Black 2017, Crispin 2008, et Gregory 2015.

2.1.2 Modèles de cycle de vie du développement logiciel en contexte

Les modèles de cycle de vie du développement logiciel doivent être sélectionnés et adaptés au contexte du projet et aux caractéristiques du produit. Un modèle de cycle de vie du développement logiciel

approprié devrait être choisi et adapté en fonction de l'objectif du projet, du type de produit développé, des priorités métier (p. ex., délai de mise sur le marché) et des risques produit et projet identifiés. Par exemple, le développement et le test d'un système peu important de gestion interne devrait être différent du développement et du test d'un système critique pour la sécurité, comme un système de contrôle de freinage d'une automobile. Par ailleurs, dans certains cas, des problèmes organisationnels et culturels peuvent inhiber la communication entre les membres de l'équipe, ce qui peut gêner le développement itératif.

Selon le contexte du projet, il peut être nécessaire de combiner ou de réorganiser les niveaux de test et/ou les activités de test. Par exemple, pour l'intégration d'un logiciel commercial sur étagère (COTS) dans un système plus grand, l'acheteur peut effectuer des tests d'interopérabilité au niveau des tests d'intégration du système (p. ex., intégration à l'infrastructure et à d'autres systèmes) et au niveau des tests d'acceptation (fonctionnels et non-fonctionnels, ainsi que des tests d'acceptation utilisateur et des tests d'acceptation opérationnelle). Voir la section 2.2 pour une description des niveaux de test et la section 2.3 pour une description des types de test.

En outre, les modèles de cycle de vie du développement logiciel eux-mêmes peuvent être combinés. Par exemple, un modèle en V peut être utilisé pour le développement et le test des systèmes back-end et de leurs intégrations, tandis qu'un modèle de développement Agile peut être utilisé pour développer et tester l'interface utilisateur front-end (IHM) et les fonctionnalités. Le prototypage peut être utilisé au début d'un projet, avec un modèle de développement incrémental adopté une fois la phase expérimentale terminée.

Les systèmes Internet des Objets (IoT – Internet of Things), qui se composent de nombreux objets différents, tels que des équipements, des produits et des services, appliquent généralement des modèles de cycle de développement logiciel distincts pour chaque objet. Cela représente un défi particulier pour le développement de versions du système global IoT. De plus, le cycle de développement logiciel de ces objets met davantage l'accent sur les phases ultérieures du cycle de vie du développement logiciel après leur mise en service (p. ex. phases d'exploitation, de mise à jour et de décommissionnement).

2.2 Niveaux de test

Les niveaux de test sont des groupes d'activités de test qui sont organisées et gérées ensemble. Chaque niveau de test est une instance du processus de test, constitué des activités décrites à la section 1.4, réalisées en relation avec le logiciel à un niveau de développement donné, depuis les unités ou composants individuels jusqu'aux systèmes complets ou, le cas échéant, aux systèmes de systèmes. Les niveaux de test sont liés à d'autres activités du cycle de vie du développement logiciel. Les niveaux de test utilisés dans ce syllabus sont les suivants :

- Test de composants
- Test d'intégration
- Test système
- Test d'acceptation

Les niveaux de test sont caractérisés par les éléments suivants :

- Objectifs spécifiques
- Bases de test, référencées pour en dériver des cas de test
- Objet de test (c.-à-d. ce qui est testé)
- Défauts et défaillances types

- Approches et responsabilités spécifiques

Pour chaque niveau de test, un environnement de test approprié est requis. Dans les tests d'acceptation, par exemple, un environnement de test représentatif de la production est idéal, tandis que dans les tests de composants, les développeurs utilisent généralement leur propre environnement de développement.

2.2.1 Test de composants

Objectifs des tests de composants

Les tests de composants (également connus sous le nom de tests unitaires ou de modules) se concentrent sur des composants qui peuvent être testés séparément. Les objectifs des tests de composants sont les suivants :

- Réduire le risque
- Vérifier si les comportements fonctionnels et non-fonctionnels du composant sont tels qu'ils ont été conçus et spécifiés
- Renforcer la confiance dans la qualité du composant
- Trouver des défauts dans le composant
- Empêcher les défauts de passer à des niveaux de test plus élevés

Dans certains cas, particulièrement dans les modèles de développement incrémentaux et itératifs (p. ex. Agile) où les changements de code sont continus, les tests de régression automatisés jouent un rôle clé dans l'établissement de la confiance que les changements n'ont pas endommagés les composants existants.

Les tests de composants sont souvent effectués indépendamment du reste du système, en fonction du modèle de cycle de développement logiciel et du système, ce qui peut nécessiter de mettre en place des objets fictifs, une virtualisation des services, des harnais de test, des bouchons et des pilotes. Les tests de composants peuvent porter sur des caractéristiques fonctionnelles (p. ex., exactitude des calculs), non-fonctionnelles (p. ex. recherche de fuites mémoire) et les propriétés structurelles (p. ex., tests des décisions).

Bases de test

Voici des exemples de produits d'activités qui peuvent être utilisés comme bases de test pour les tests de composants :

- Conception détaillée
- Code
- Modèle de données
- Spécifications des composants

Objets de test

Les objets de test habituels pour les tests de composants sont les suivants :

- Composants, unités ou modules
- Code et structures de données
- Classes

- Modules de bases de données

Défauts et défaillances courants

Voici des exemples de défauts et de défaillances courants pour les tests de composants :

- Fonctionnalité incorrecte (par exemple, non conforme aux spécifications de conception)
- Problèmes de flux de données
- Code et logique incorrects

Les défauts sont généralement corrigés dès qu'ils sont détectés, souvent sans gestion formelle des défauts. Cependant, lorsque les développeurs rapportent vraiment des défauts, cela donne des informations importantes pour l'analyse des causes racines et l'amélioration des processus.

Approches spécifiques et responsabilités

Les tests de composants sont généralement effectués par le développeur qui a écrit le code, mais il nécessite au minimum l'accès au code testé. Les développeurs peuvent alterner le développement des composants avec la recherche et la correction de défauts. Les développeurs vont souvent écrire et exécuter des tests après avoir écrit le code d'un composant. Cependant, dans le développement Agile en particulier, l'écriture de cas de test de composants automatisés peut précéder l'écriture du code de l'application.

Considérons par exemple le développement piloté par les tests (TDD : Test Driven Development). Le développement piloté par les tests est fortement itératif et basé sur des cycles de développement de cas de tests automatisés, puis la construction et l'intégration de petits morceaux de code, suivi de l'exécution des tests des composants, la correction de tout problème et le refactoring du code. Ce processus se poursuit jusqu'à ce que le composant soit complètement achevé et que tous les tests de composants réussissent. Le développement piloté par les tests est un exemple de l'approche dite « test-first » - tester en premier. Le développement piloté par les tests a commencé avec l'eXtreme Programming (XP) et il s'est étendu à d'autres formes de développement Agile et aussi à des cycles de vie séquentiels (cf. syllabus Testeur Certifié - Extension Niveau Fondation Testeur Agile CFTL / ISTQB).

2.2.2 Test d'intégration

Objectifs des tests d'intégration

Les tests d'intégration se concentrent sur les interactions entre les composants ou les systèmes. Les objectifs des tests d'intégration comprennent :

- Réduire les risques
- Vérifier si les comportements fonctionnels et non-fonctionnels des interfaces sont tels qu'ils ont été conçus et spécifiés
- Renforcer la confiance dans la qualité des interfaces
- Trouver des défauts (qui peuvent se trouver dans les interfaces elles-mêmes ou dans les composants ou systèmes)
- Empêcher les défauts de passer à des niveaux de test plus élevés

Comme pour les tests de composants, dans certains cas, les tests d'intégration automatisés utilisés en tests de régression donnent l'assurance que les changements n'ont pas altéré les interfaces, composants ou systèmes existants.

Il y a deux niveaux différents de tests d'intégration décrits dans ce syllabus, qui peuvent être effectués sur des objets de test de taille variable comme suit :

- Les tests d'intégration de composants se concentrent sur les interactions et les interfaces entre composants intégrés. Les tests d'intégration de composants sont effectués après les tests de composants et sont généralement automatisés. Dans le développement itératif et incrémental, les tests d'intégration des composants font généralement partie du processus d'intégration continue.
- Les tests d'intégration de systèmes se concentrent sur les interactions et les interfaces entre systèmes, progiciels et micro-services. Les tests d'intégration de systèmes peuvent également couvrir les interactions avec des entités externes (p. ex. services Web) et les interfaces fournies par celles-ci. Dans ce cas, l'organisation qui développe ne contrôle pas les interfaces externes, ce qui peut créer divers problèmes pour les tests (p. ex. s'assurer que des défauts bloquants pour les tests dans le code de l'entité externe sont résolus, organiser des environnements de test, etc.). Les tests d'intégration de systèmes peuvent être effectués après les tests systèmes ou en parallèle d'activités de tests systèmes en cours (à la fois en développement séquentiel et en développement itératif et incrémental).

Bases de test

Voici des exemples de produits d'activités qui peuvent servir de bases de test pour les tests d'intégration :

- Conception du logiciel et du système
- Diagrammes de séquence
- Spécifications des protocoles d'interface et de communication
- Cas d'utilisation
- Architecture au niveau du composant ou du système
- Workflows
- Définitions des interfaces externes

Objets de test

Les objets de test habituels pour les tests d'intégration comprennent :

- Sous-systèmes
- Bases de données
- Infrastructure
- Interfaces
- APIs
- Micro-services

Défauts et défaillances courants

Voici des exemples de défauts et de défaillances habituels pour les tests d'intégration de composants :

- Données incorrectes, données manquantes ou encodage incorrect des données
- Mauvais séquencement ou synchronisation des appels d'interfaces
- Décalages au niveau des interfaces

- Défaillances dans la communication entre les composants
- Défaillances de communication non gérées ou mal gérées entre les composants
- Hypothèses incorrectes sur la signification, les unités ou les limites des données transmises entre les composants

Voici des exemples de défauts et de défaillances habituels pour les tests d'intégration de systèmes :

- Structures de message incohérentes entre les systèmes
- Données incorrectes, données manquantes ou encodage incorrect des données
- Décalages au niveau des interfaces
- Défaillances dans la communication entre les systèmes
- Défaillances de communication non gérées ou mal gérées entre les systèmes
- Hypothèses incorrectes sur la signification, les unités ou les limites des données transmises entre les systèmes
- Non-respect des règles de sécurité requises

Approches spécifiques et responsabilités

Les tests d'intégration de composants et les tests d'intégration de systèmes doivent se concentrer sur l'intégration elle-même. Par exemple, pour l'intégration du module A avec le module B, les tests devraient se concentrer sur la communication entre les modules, et non sur la fonctionnalité des modules individuels, car cela aurait dû être couvert pendant les tests de composants. Pour l'intégration du système X au système Y, les tests devraient se concentrer sur la communication entre les systèmes, et non sur la fonctionnalité des systèmes individuels, car cela aurait dû être couvert pendant les tests du système. Les types de tests fonctionnels, non-fonctionnels et structurels sont concernés.

Les tests d'intégration de composants sont souvent la responsabilité des développeurs. Les tests d'intégration de systèmes relèvent généralement de la responsabilité des testeurs. Idéalement, les testeurs qui effectuent des tests d'intégration de systèmes devraient comprendre l'architecture du système et devraient avoir contribué à la planification de l'intégration.

Si les tests d'intégration et la stratégie d'intégration sont planifiés avant le développement des composants ou des systèmes, ces composants ou systèmes peuvent être construits dans l'ordre requis pour des tests plus efficaces. Les stratégies d'intégration systématique peuvent être fondées sur l'architecture du système (p. ex. de haut en bas et de bas en haut), les tâches fonctionnelles, les séquences de traitement des transactions ou d'autres aspects du système ou des composantes. Afin de simplifier l'isolation des défauts et de détecter tôt les défauts, l'intégration devrait normalement être incrémentale (c'est-à-dire un petit nombre de composants ou de systèmes supplémentaires à la fois) plutôt que "big bang" (c'est-à-dire l'intégration de tous les composants ou systèmes en une seule étape). Une analyse de risque des interfaces les plus complexes peut aider à focaliser les tests d'intégration.

Plus la portée de l'intégration est grande, plus il devient difficile de localiser les défauts d'un composant ou d'un système spécifique, ce qui peut entraîner un risque accru et du temps supplémentaire pour la correction des problèmes rencontrés. C'est l'une des raisons pour lesquelles l'intégration continue, où le logiciel est intégré composant par composant (c'est-à-dire l'intégration fonctionnelle), est devenue une pratique courante. Une telle intégration continue inclut souvent des tests de régression automatisés, idéalement à plusieurs niveaux de test.

2.2.3 Test système

Objectifs des tests système

Les tests système se concentrent sur le comportement et les capacités d'un système ou d'un produit entier, en considérant souvent les tâches de bout en bout que le système peut exécuter et les comportements non-fonctionnels qu'il présente pendant l'exécution de ces tâches. Les objectifs des tests système comprennent :

- Réduire les risques
- Vérifier si les comportements fonctionnels et non-fonctionnels du système sont tels qu'ils ont été conçus et spécifiés
- Valider que le système est complet et fonctionnera comme prévu
- Renforcer la confiance dans la qualité du système dans son ensemble
- Trouver des défauts
- Empêcher les défauts de passer à des niveaux de test plus élevés ou en production

Pour certains systèmes, la vérification de la qualité des données peut être un objectif. Comme pour les tests de composants et les tests d'intégration, dans certains cas, les tests de régression automatisés au niveau système donnent l'assurance que les changements n'ont pas altéré les caractéristiques existantes ou les capacités de bout en bout. Le test système produit souvent de l'information qui est utilisée par les intervenants pour prendre des décisions de livraison. Les tests systèmes peuvent également satisfaire aux exigences ou aux normes légales ou réglementaires.

L'environnement de test devrait idéalement correspondre à la cible finale ou à l'environnement de production.

Bases de test

Voici des exemples de produits d'activités qui peuvent être utilisés comme bases de test pour les tests système :

- Spécifications des exigences système et logicielles (fonctionnelles et non-fonctionnelles)
- Rapports d'analyse des risques
- Cas d'utilisation
- Epics et User Stories
- Modèles de comportement du système
- Diagrammes d'états
- Manuels système et manuels d'utilisation

Objets de test

Les objets de test habituels pour les tests système comprennent :

- Applications
- Systèmes Matériel/Logiciel
- Systèmes d'exploitation

- Système sous test (SUT – System Under Test)
- Configuration du système et données de configuration

Défauts et défaillances courants

Voici des exemples de défauts et de défaillances habituels pour les tests système :

- Calculs incorrects
- Comportement fonctionnel ou non-fonctionnel du système incorrect ou inattendu
- Flux de contrôle et/ou de données incorrects au sein du système
- Réalisation incorrecte et incomplète des tâches fonctionnelles de bout en bout
- Incapacité du système à fonctionner correctement dans le ou les environnements de production
- Incapacité du système à fonctionner selon la description faite dans les manuels système et utilisateur

Approches spécifiques et responsabilités

Les tests système devraient se concentrer sur le comportement global de bout en bout du système dans son ensemble, à la fois fonctionnel et non-fonctionnel. Les tests système doivent utiliser les techniques de test les plus appropriées (voir chapitre 4) pour l'(les) aspect(s) du système à tester. Par exemple, une table de décision peut être créée pour vérifier si le comportement fonctionnel est tel que décrit dans les règles métier.

Des testeurs indépendants procèdent en général aux tests système. Des défauts dans les spécifications (par exemple, des User Stories manquantes, des exigences métier mal énoncées, etc.) peuvent conduire à un manque de compréhension ou à des désaccords sur le comportement attendu du système. De telles situations peuvent causer des faux positifs et des faux négatifs, ce qui fait perdre du temps et réduit l'efficacité de la détection des défauts. L'implication en amont des testeurs dans l'affinement des User Stories ou dans les activités de tests statiques, comme les revues, aide à réduire l'incidence de telles situations.

2.2.4 Test d'acceptation

Objectifs des tests d'acceptation

Les tests d'acceptation, comme les tests système, se concentrent généralement sur le comportement et les capacités d'un système ou d'un produit entier. Les objectifs des tests d'acceptation comprennent :

- Établir la confiance dans la qualité du système dans son ensemble
- Valider que le système est complet et qu'il fonctionnera comme attendu
- Vérifier que les comportements fonctionnels et non-fonctionnels du système sont tels que spécifiés

Les tests d'acceptation peuvent produire des informations permettant d'évaluer la disponibilité du système pour le déploiement et l'utilisation par le client (utilisateur final). Des défauts peuvent être trouvés pendant les tests d'acceptation, mais la découverte de défauts ne constitue en général pas un objectif, et la découverte d'un nombre significatif de défauts pendant les tests d'acceptation peut dans certains cas être considérée comme un risque majeur du projet. Les tests d'acceptation peuvent également satisfaire aux exigences ou aux normes légales ou réglementaires.

Les formes communes de tests d'acceptation comprennent :

- Tests d'acceptation utilisateur
- Tests d'acceptation opérationnelle
- Tests d'acceptation contractuelle et réglementaire
- Tests alpha et bêta

Chacune est décrite dans les quatre sous-sections suivantes.

Tests d'acceptation utilisateur (UAT - User Acceptance Testing)

Les tests d'acceptation du système par les utilisateurs sont généralement axés sur la validation de son aptitude à l'usage par les utilisateurs prévus dans un environnement opérationnel réel ou simulé. L'objectif principal est de renforcer la confiance dans l'usage du système pour répondre aux besoins des utilisateurs, satisfaire les exigences et exécuter les processus métier avec un minimum de difficultés, de coûts et de risques.

Tests d'acceptation opérationnelle (OAT - Operational Acceptance Testing)

Les tests d'acceptation du système par le service d'exploitation ou le service d'administration du système sont généralement effectués dans un environnement de production (simulé). Les tests se concentrent sur les aspects opérationnels et peuvent inclure les éléments suivants :

- Tests de sauvegarde et de restauration
- Installation, désinstallation et mise à jour
- Reprise après sinistre
- Gestion des utilisateurs
- Tâches de maintenance
- Tâches de chargement et de migration des données
- Vérification des vulnérabilités de sécurité
- Tests de performance

L'objectif principal des tests d'acceptation opérationnelle est de renforcer la confiance dans le fait que les opérateurs ou les administrateurs système peuvent maintenir le système en bon état de fonctionnement pour les utilisateurs dans l'environnement opérationnel, même dans des conditions exceptionnelles ou difficiles.

Tests d'acceptation contractuelle et réglementaire

Les tests d'acceptation contractuelle sont effectués en fonction des critères d'acceptation d'un contrat pour la production de logiciels sur mesure. Les critères d'acceptation devraient être définis lorsque les parties conviennent du contrat. Les tests d'acceptation contractuelle sont souvent effectués par les utilisateurs ou par des testeurs indépendants.

Les tests d'acceptation réglementaire sont effectués par rapport à tous les règlements qui doivent être respectés, comme les règlements gouvernementaux, légaux ou de sécurité. Les tests d'acceptation réglementaire sont souvent effectués par les utilisateurs ou par des testeurs indépendants, les résultats étant parfois observés ou vérifiés par des organismes de réglementation.

L'objectif principal des tests d'acceptation contractuelle et réglementaire est de renforcer la confiance dans l'atteinte de la conformité contractuelle ou réglementaire.

Tests alpha et bêta

Les tests alpha et bêta sont généralement utilisés par les développeurs de logiciels commerciaux sur étagère (COTS) qui souhaitent obtenir une évaluation des utilisateurs, clients et/ou opérateurs potentiels ou existants avant que le produit logiciel ne soit mis sur le marché. Les tests alpha sont effectués sur le site de l'organisation qui développe, non pas par l'équipe de développement, mais par des clients potentiels ou existants, et/ou des opérateurs ou une équipe de test indépendante. Les tests bêta sont effectués par des clients potentiels ou existants, et/ou des opérateurs sur leurs propres sites. Le test bêta peut venir après le test alpha, ou peut se produire sans qu'aucun test alpha précédent n'ait eu lieu.

L'un des objectifs des tests alpha et bêta est de renforcer la confiance des clients potentiels ou existants et/ou des opérateurs dans l'utilisation du système dans des conditions normales et quotidiennes et dans l'environnement opérationnel pour atteindre leurs objectifs avec un minimum de difficulté, de coût et de risque. Un autre objectif peut être la détection des défauts liés aux conditions et aux environnements dans lesquels le système sera utilisé, en particulier lorsque ces conditions et environnements sont difficiles à reproduire par l'équipe de développement.

Bases de test

Voici des exemples de produits d'activités qui peuvent être utilisés comme base de test pour toute forme de test d'acceptation :

- Processus métier
- Exigences utilisateur ou métier
- Réglementations, contrats légaux et normes
- Cas d'utilisation
- Exigences système
- Documentation système ou utilisateur
- Procédures d'installation
- Rapports d'analyse des risques

En outre, comme base de test pour dériver des cas de test pour les tests d'acceptation opérationnelle, un ou plusieurs des produits d'activités suivants peuvent être utilisés :

- Procédures de sauvegarde et de restauration
- Procédures de reprise après sinistre
- Exigences non-fonctionnelles
- Documentation d'exploitation
- Instructions de déploiement et d'installation
- Objectifs de performance
- Ensembles de base de données
- Normes ou règlements de sécurité

Objets de test habituels

Les objets de test habituels pour toute forme de test d'acceptation sont les suivants :

- Systèmes sous test
- Configuration du système et données de configuration
- Processus métier pour un système entièrement intégré
- Systèmes de récupération et sites sensibles (pour les tests de continuité d'activité et de reprise après sinistre)
- Processus d'exploitation et de maintenance
- Formulaires
- Rapports
- Données de production existantes et modifiées

Défauts et défaillances courants

Voici des exemples de défauts courants pour toute forme de test d'acceptation :

- Les workflows du système ne répondent pas aux exigences métier ou utilisateurs
- Les règles métier ne sont pas correctement implémentées
- Le système ne satisfait pas aux exigences contractuelles ou réglementaires
- Les défaillances non-fonctionnelles telles que les vulnérabilités de sécurité, le manque de performance en cas de charges élevées, ou le mauvais fonctionnement sur une plate-forme supportée

Approches et responsabilités spécifiques

Les tests d'acceptation relèvent souvent de la responsabilité des clients, des utilisateurs métier, des Product Owners ou des exploitants d'un système, et d'autres parties prenantes pouvant également être impliquées.

Le test d'acceptation est souvent considéré comme le dernier niveau de test dans un cycle de vie de développement séquentiel, mais il peut aussi se produire à d'autres moments, par exemple :

- Les tests d'acceptation d'un produit logiciel COTS peuvent se produire lorsqu'il est installé ou intégré
- Le test d'acceptation d'une nouvelle amélioration fonctionnelle peut avoir lieu avant le test système

Dans le développement itératif, les équipes de projet peuvent employer diverses formes de tests d'acceptation pendant et à la fin de chaque itération, telles que celles qui visent à vérifier une nouvelle fonctionnalité par rapport à ses critères d'acceptation et celles qui visent à valider qu'une nouvelle fonctionnalité satisfait les besoins des utilisateurs. De plus, des tests alpha et bêta peuvent être réalisés, soit à la fin de chaque itération, après l'achèvement de chaque itération, soit après une série d'itérations. Les tests d'acceptation utilisateur, les tests d'acceptation opérationnelle, les tests d'acceptation réglementaire et les tests d'acceptation contractuelle peuvent également avoir lieu, soit à la clôture de chaque itération, après l'achèvement de chaque itération, soit après une série d'itérations.

2.3 Types de test

Un type de test est un groupe d'activités de test visant à tester des caractéristiques spécifiques d'un système logiciel ou d'une partie d'un système, sur la base d'objectifs de test spécifiques. Ces objectifs peuvent inclure :

- Évaluer des caractéristiques de qualité fonctionnelle, telles que la complétude, l'exactitude et la pertinence
- Évaluer des caractéristiques de qualité non-fonctionnelles, telles que la fiabilité, la performance, la sécurité, la compatibilité et la facilité d'utilisation
- Évaluer si la structure ou l'architecture du composant ou du système est correcte, complète et conforme aux spécifications
- Évaluer les effets des changements, tels que la confirmation de la correction de défauts (tests de confirmation) et la recherche de changements non désirés résultant de changements dans le logiciel ou l'environnement (tests de régression)

2.3.1 Tests fonctionnels

Les tests fonctionnels d'un système impliquent des tests qui évaluent les fonctionnalités que le système devrait réaliser. Les exigences fonctionnelles peuvent être décrites dans des produits d'activités tels que les spécifications des exigences métier, les épics, les User Stories, les cas d'utilisation ou les spécifications fonctionnelles, ou elles peuvent ne pas être documentées. Les fonctionnalités sont "ce que" le système doit faire.

Les tests fonctionnels devraient être effectués à tous les niveaux de test (par exemple, les tests de composants peuvent être basés sur une spécification de composant), bien que la focalisation soit différente à chaque niveau (voir section 2.2).

Les tests fonctionnels prennent en compte le comportement du logiciel, de sorte que des techniques boîte-noire peuvent être utilisées pour dériver des conditions de test et des cas de test pour la fonctionnalité du composant ou du système (voir section 4.2).

La complétude des tests fonctionnels peut être mesurée par la couverture fonctionnelle. La couverture fonctionnelle est la mesure selon laquelle un certain type d'élément fonctionnel a été exercé par des tests. Elle est exprimée en pourcentage du ou des types d'éléments couverts. Par exemple, en utilisant la traçabilité entre les tests et les exigences fonctionnelles, le pourcentage de ces exigences qui sont couvertes par les tests peut être calculé, identifiant potentiellement des lacunes de couverture.

La conception et l'exécution des tests fonctionnels peuvent nécessiter des compétences ou des connaissances particulières, comme la connaissance du problème métier particulier que le logiciel résout (par exemple, un logiciel de modélisation géologique pour les industries pétrolière et gazière) ou le rôle particulier que joue le logiciel (par exemple, un logiciel de jeux vidéo qui fournit des loisirs interactifs).

2.3.2 Tests non-fonctionnels

Les tests non-fonctionnels d'un système évaluent les caractéristiques des systèmes et des logiciels comme l'utilisabilité, la performance ou la sécurité. Il convient de se reporter à la norme ISO (ISO/CEI 25010) pour une classification des caractéristiques de qualité des produits logiciels. Le test non-fonctionnel est le test de "comment" le système se comporte.

Contrairement aux idées fausses courantes, les tests non-fonctionnels peuvent et devraient souvent être effectués à tous les niveaux de test, et ce, le plus tôt possible. La découverte tardive de défauts non-fonctionnels peut être extrêmement préjudiciable à la réussite d'un projet.

Des techniques de boîte-noire (voir section 4.2) peuvent être utilisées pour dériver des conditions de test et des cas de test pour les tests non-fonctionnels. Par exemple, l'analyse des valeurs limites peut être utilisée pour définir les conditions de sollicitation pour les tests de performance.

La complétude des tests non-fonctionnels peut être mesurée par une couverture non-fonctionnelle. La couverture non-fonctionnelle est la mesure selon laquelle un certain type d'élément non-fonctionnel a été exercé par des tests et est exprimée en pourcentage du ou des types d'éléments couverts. Par exemple, en utilisant la traçabilité entre les tests et les appareils pris en charge pour une application mobile, le pourcentage d'appareils qui font l'objet de tests de compatibilité peut être calculé, identifiant potentiellement les lacunes de couverture.

La conception et l'exécution de tests non-fonctionnels peuvent impliquer des compétences ou des connaissances spécifiques, telles que la connaissance des faiblesses inhérentes à une conception ou à une technologie (par exemple, les vulnérabilités de sécurité associées à des langages de programmation particuliers) ou la spécificité des utilisateurs (p. ex. les types d'utilisateurs des systèmes de gestion des établissements de santé).

Se référer aux syllabus CFTL / ISTQB Testeur Certifié de Niveau Avancé : Analyste de Test, Analyste Technique de Test, Testeur de Sécurité, et d'autres modules spécialisés du CFTL / ISTQB pour plus de détails concernant les tests des caractéristiques de qualité non-fonctionnelles.

2.3.3 Tests boîte-blanche

Les tests boîte-blanche sont des tests basés sur la structure ou l'implémentation interne du système. La structure interne peut comprendre le code, l'architecture, les flux de travail et/ou les flux de données au sein du système (voir section 4.3).

La complétude des tests boîte-blanche peut être mesurée par la couverture structurelle. La couverture structurelle est la mesure dans laquelle un certain type d'élément structurel a été exercé par des tests et est exprimée en pourcentage du type d'élément couvert.

Au niveau du test des composants, la couverture du code est basée sur le pourcentage de code du composant qui a été testé, et peut être mesurée sur différents aspects du code (éléments de couverture) tels que le pourcentage d'instructions exécutables testées dans le composant, ou le pourcentage de résultats de décision testés. Ces types de couverture sont collectivement appelés couverture de code. Au niveau des tests d'intégration des composants, les tests de boîte-blanche peuvent être basés sur l'architecture du système, comme les interfaces entre composants, et la couverture structurelle peut être mesurée en terme de pourcentage d'interfaces exercées par les tests.

La conception et l'exécution de tests boîte-blanche peuvent nécessiter des compétences ou des connaissances particulières, comme la façon dont le code est construit (p. ex. utiliser des outils de couverture de code), la façon dont les données sont stockées (p. ex. évaluer les requêtes possibles dans les bases de données) et la façon d'utiliser les outils de couverture et d'interpréter correctement leurs résultats.

2.3.4 Tests liés aux changements

Lorsque des modifications sont apportées à un système, que ce soit pour corriger un défaut ou en raison d'une fonctionnalité nouvelle ou modifiée, des tests devraient être effectués pour confirmer que les

modifications ont corrigé le défaut ou implémenté la fonctionnalité correctement, et n'ont pas causé de conséquences préjudiciables inattendues.

- Test de confirmation : Après la correction d'un défaut, le logiciel peut être testé avec tous les cas de test qui ont échoué en raison du défaut, qui doivent être ré-exécutés sur la nouvelle version du logiciel. Le logiciel peut également être testé avec de nouveaux tests si, par exemple, le défaut était une fonctionnalité manquante. A minima, les étapes pour reproduire le(s) défaut(s) causé(s) par le défaut doivent être ré-exécutées sur la nouvelle version du logiciel. Le but d'un test de confirmation est de confirmer que le défaut d'origine a été réparé avec succès.
- Tests de régression : Il est possible qu'une modification apportée à une partie du code, qu'il s'agisse d'une correction ou d'un autre type de modification, puisse accidentellement affecter le comportement d'autres parties du code, que ce soit au sein du même composant, dans d'autres composants du même système ou même dans d'autres systèmes. Les changements peuvent inclure des changements de l'environnement, comme une nouvelle version d'un système d'exploitation ou d'un système de gestion de base de données. De tels effets de bord involontaires sont appelés régressions. Les tests de régression sont des tests visant à détecter de tels effets de bord involontaires.

Des tests de confirmation et de régression sont effectués à tous les niveaux de test.

En particulier dans les cycles de vie de développement itératif et incrémental (p. ex. en Agile), les nouvelles caractéristiques, les modifications apportées aux caractéristiques existantes et le refactoring du code entraînent des changements fréquents du code, ce qui nécessite également des tests liés aux changements. En raison de la nature évolutive du système, les tests de confirmation et de régression sont très importants. Ceci est particulièrement important pour les systèmes Internet des objets où les objets individuels (p. ex. des appareils) sont fréquemment mis à jour ou remplacés.

Les suites de tests de régression sont exécutées plusieurs fois et évoluent généralement lentement, raison pour laquelle les tests de régression sont de bons candidats pour l'automatisation. L'automatisation de ces tests devrait commencer tôt dans le projet (voir chapitre 6).

2.3.5 Types de test et niveaux de test

Il est possible d'effectuer n'importe quel type de test mentionné ci-dessus à n'importe quel niveau de test. Pour illustrer, des exemples de tests fonctionnels, non-fonctionnels, boîte blanche et liés au changement seront donnés pour tous les niveaux de test, pour une application bancaire, en commençant par des tests fonctionnels :

- Pour les tests de composants, les tests sont conçus en fonction de la façon dont un composant doit calculer les intérêts composés.
- Pour les tests d'intégration de composants, les tests sont conçus en fonction de la façon dont les informations saisies au niveau de l'interface utilisateur sont transmises à la couche métier.
- Pour les tests système, les tests sont conçus en fonction de la façon dont les titulaires de comptes peuvent demander une ligne de crédit sur leurs comptes chèques.
- Pour les tests d'intégration au niveau système, les tests sont conçus en fonction de la façon dont le système utilise un micro-service externe pour vérifier le taux de crédit d'un titulaire de compte.
- Pour les tests d'acceptation, les tests sont conçus en fonction de la façon dont le banquier gère l'approbation ou le refus d'une demande de crédit.

Voici des exemples de tests non-fonctionnels :

- Pour les tests de composants, les tests de performance sont conçus pour évaluer le nombre de cycles CPU nécessaires pour effectuer un calcul d'intérêt total complexe.
- Pour les tests d'intégration de composants, les tests de sécurité sont conçus pour les vulnérabilités de débordement de la mémoire tampon dues aux données transmises de l'interface utilisateur à la couche métier.
- Pour les tests système, les tests de portabilité sont conçus pour vérifier si la couche de présentation fonctionne sur tous les navigateurs et appareils mobiles supportés.
- Pour les tests d'intégration de système, les tests de fiabilité sont conçus pour évaluer la robustesse du système lorsque le micro-service de calcul du taux de crédit ne répond pas.
- Pour les tests d'acceptation, les tests d'utilisabilité sont conçus pour évaluer l'accessibilité de l'interface de traitement du crédit du banquier pour les personnes handicapées.

Voici des exemples de tests boîte-blanche :

- Pour les tests de composants, les tests sont conçus pour obtenir une couverture complète des instructions et des décisions (voir section 4.3) pour tous les composants qui effectuent des calculs financiers.
- Pour les tests d'intégration de composants, les tests sont conçus pour vérifier comment chaque écran de l'interface du navigateur transmet les données à l'écran suivant et à la couche métier.
- Pour les tests système, les tests sont conçus pour couvrir les séquences de pages Web qui peuvent se succéder pendant une demande de ligne de crédit.
- Pour les tests d'intégration de système, les tests sont conçus pour exercer tous les types de requête possibles envoyés au micro-service de calcul du taux de crédit.
- Pour les tests d'acceptation, les tests sont conçus pour couvrir toutes les structures de fichiers de données financières prises en charge et les plages de valeurs pour les transferts de banque à banque.

Enfin, voici des exemples de tests liés aux changements :

- Pour les tests de composants, des tests de régression automatisés sont construits pour chaque composant et inclus dans le framework d'intégration continue.
- Pour les tests d'intégration de composants, les tests sont conçus pour confirmer les corrections des défauts liés à l'interface au fur et à mesure que ces corrections sont intégrées dans le référentiel de code.
- Pour les tests système, tous les tests d'un workflow donné sont ré-exécutés si un écran de ce workflow change.
- Pour les tests d'intégration système, les tests de l'application interagissant avec le micro-service de calcul du taux de crédit sont ré-exécutés quotidiennement dans le cadre du déploiement continu de ce micro-service.
- Pour les tests d'acceptation, tous les tests échoués précédemment sont ré-exécutés après la correction d'un défaut constaté lors des tests d'acceptation.

Bien que cette section donne des exemples de chaque type de test à tous les niveaux, il n'est pas nécessaire, pour tous les logiciels, que chaque type de test soit représenté à tous les niveaux. Toutefois, il est important d'exécuter les types de test applicables à chaque niveau, en particulier au niveau le plus précoce où le type de test se trouve.

2.4 Tests de maintenance

Une fois déployés dans les environnements de production, les logiciels et les systèmes doivent être maintenus. Des changements de diverses sortes sont presque inévitables dans les logiciels et les systèmes livrés, soit pour corriger des défauts découverts lors de l'utilisation opérationnelle, soit pour ajouter de nouvelles fonctionnalités, soit pour supprimer ou modifier des fonctionnalités déjà livrées. La maintenance est également nécessaire pour préserver ou améliorer les caractéristiques de qualité non-fonctionnelles du composant ou du système pendant toute sa durée de vie, en particulier la performance, la compatibilité, la fiabilité, la sécurité et la portabilité.

Lorsque des modifications sont apportées dans le cadre de la maintenance, des tests de maintenance devraient être effectués, à la fois pour évaluer le succès avec lequel les modifications ont été apportées et pour vérifier les effets secondaires possibles (p. ex. régressions) dans les parties du système qui demeurent inchangées (ce qui est habituellement la plus grande partie du système). Les tests de maintenance se concentrent sur le test des changements apportés au système, ainsi que sur le test des parties inchangées qui auraient pu être affectées par les changements. La maintenance peut impliquer des versions planifiées et des versions non planifiées (corrections à chaud).

Une version de maintenance peut nécessiter des tests de maintenance à plusieurs niveaux de test, en utilisant différents types de test, en fonction de sa portée. L'étendue des tests de maintenance dépend des éléments suivants :

- Le degré de risque du changement, par exemple, la mesure dans laquelle le périmètre modifié du logiciel communique avec d'autres composants ou systèmes
- La taille du système existant
- La taille du changement

2.4.1 Facteurs déclencheurs pour la maintenance

Il y a plusieurs raisons pour lesquelles la maintenance logicielle, et donc les tests de maintenance, ont lieu, à la fois pour les changements planifiés et non planifiés.

Nous pouvons classer les facteurs déclencheurs de la maintenance comme suit :

- Modification, comme des améliorations planifiées (p. ex., basées sur les versions), des changements correctifs et d'urgence, des changements de l'environnement opérationnel (comme des mises à niveau planifiées du système d'exploitation ou de la base de données), des mises à niveau de logiciels sur étagère (COTS) et des correctifs pour les défauts et les vulnérabilités
- Migration, par exemple d'une plate-forme à une autre, qui peut nécessiter des tests opérationnels du nouvel environnement ainsi que du logiciel modifié, ou des tests de conversion de données lorsque les données d'une autre application seront migrées dans le système en cours de maintenance
- Déclassement, par exemple lorsqu'une application arrive en fin de vie

Lorsqu'une application ou un système est mis hors service, il peut être nécessaire de tester la migration ou l'archivage des données si de longues périodes de conservation des données sont nécessaires. Il peut également être nécessaire de tester les procédures de restauration/récupération après l'archivage pour de longues périodes de conservation. En outre, des tests de régression peuvent être nécessaires pour s'assurer que toute fonctionnalité qui reste en service fonctionne toujours.

Pour les systèmes Internet des objets, les tests de maintenance peuvent être déclenchés par l'introduction d'objets complètement nouveaux ou modifiés, tels que des dispositifs matériels et des

services logiciels, dans l'ensemble du système. Les tests de maintenance de ces systèmes mettent particulièrement l'accent sur les tests d'intégration à différents niveaux (par exemple, niveau réseau, niveau application) et sur les aspects de sécurité, en particulier ceux relatifs aux données personnelles.

2.4.2 Analyse d'impact pour la maintenance

L'analyse d'impact évalue les changements qui ont été apportés pour une version de maintenance afin d'identifier les conséquences prévues ainsi que des effets secondaires attendus et possibles d'un changement, et d'identifier les zones du système qui seront affectées par le changement. L'analyse d'impact peut également aider à identifier l'impact d'un changement sur les tests existants. Les effets secondaires et les zones affectées dans le système doivent être testés pour les régressions, éventuellement après la mise à jour des tests existants affectés par le changement.

L'analyse d'impact peut être effectuée avant qu'un changement ne soit apporté, pour aider à déterminer si le changement devrait être apporté en fonction des conséquences potentielles dans d'autres parties du système.

L'analyse d'impact peut être difficile si :

- Les spécifications (p. ex., exigences métier, User Stories, architecture) sont obsolètes ou manquantes
- Les cas de test ne sont pas documentés ou sont obsolètes
- La traçabilité bidirectionnelle entre les tests et les bases de test n'a pas été maintenue
- Le support de l'outillage est faible ou inexistant
- Les personnes impliquées n'ont pas de connaissance du domaine et/ou du système
- Une attention insuffisante a été accordée à la maintenabilité du logiciel au cours du développement

3 Tests statiques — 135 minutes

Termes

revue ad hoc, revue basée sur des checklists, test dynamique, revue formelle, revue informelle, inspection, relecture basée sur la perspective, revue, revue basée sur les rôles, revue basée sur des scénarios, analyse statique, test statique, revue technique, relecture technique

Objectifs d'apprentissage pour les tests statiques

3.1 Bases des tests statiques

FL-3.1.1 (K1) Identifier les types de produits d'activité logiciels qui peuvent être examinés par les différentes techniques de test statique

FL-3.1.2 (K2) Utiliser des exemples pour décrire la valeur du test statique

FL-3.1.3 (K2) Expliquer la différence entre les techniques statiques et dynamiques, en considérant les objectifs, les types de défauts à identifier et le rôle de ces techniques dans le cycle de vie du logiciel

3.2 Processus de revue

FL-3.2.1 (K2) Résumer les activités du processus de revue de produits d'activité

FL-3.2.2 (K1) Identifier les différents rôles et responsabilités d'une revue formelle

FL-3.2.3 (K2) Expliquer les différences entre les différents types de revue : revue informelle, relecture technique, revue technique, et inspection

FL-3.2.4 (K3) Appliquer une technique de revue sur un produit d'activités afin d'y trouver des défauts

FL-3.2.5 (K2) Expliquer les facteurs contribuant au succès des revues

3.1 Bases des tests statiques

Contrairement aux tests dynamiques, qui nécessitent l'exécution du logiciel testé, les tests statiques reposent sur l'examen manuel des produits d'activités (c.-à-d. les revues) ou sur une évaluation outillée du code ou d'autres produits d'activités (c.-à-d. l'analyse statique). Les deux types de tests statiques évaluent le code ou tout autre produit d'activités testé sans exécuter réellement le code ou le produit d'activités testé.

L'analyse statique est importante pour les systèmes informatiques critiques pour la sûreté (p. ex. logiciels aéronautiques, médicaux ou nucléaires), mais l'analyse statique est également devenue importante et courante dans d'autres contextes. Par exemple, l'analyse statique est une partie importante des tests de sécurité. L'analyse statique est aussi souvent incorporée dans les systèmes automatisés de build et de livraison, par exemple dans le développement Agile, en livraison en continu et en déploiement continu.

3.1.1 Produits d'activités qui peuvent être examinés par des tests statiques

Presque tous les produits d'activités peuvent être examinés à l'aide de tests statiques (revues et/ou analyses statiques), par exemple :

- Les spécifications, y compris les exigences métier, les exigences fonctionnelles et les exigences de sécurité
- Les épics, User Stories, et critères d'acceptation
- Les spécifications d'architecture et de conception
- Le code
- Le testware, y compris les plans de test, les cas de test, les procédures de test et les scripts de test automatisés
- Les guides utilisateur
- Les pages Web
- Les contrats, les plans de projet, les calendriers et les budgets
- Les modèles, tels que les diagrammes d'activité, qui peuvent être utilisés pour les tests basés sur des modèles (cf. syllabus Model-Based Testing, Niveau Fondation CFTL / ISTQB et Kramer 2016)

Les revues peuvent être appliquées à n'importe quel produit d'activités que les participants peuvent lire et comprendre. L'analyse statique peut être appliquée efficacement à tout produit d'activités ayant une structure formelle (généralement un code ou des modèles) pour lequel il existe un outil d'analyse statique approprié. L'analyse statique peut même être appliquée avec des outils qui évaluent les produits d'activités écrits en langage naturel comme les exigences (par exemple, vérification de l'orthographe, de la grammaire et de la lisibilité).

3.1.2 Bénéfices des tests statiques

Les techniques de tests statiques apportent plusieurs bénéfices. Lorsqu'ils sont appliqués tôt dans le cycle de vie du développement du logiciel, les tests statiques permettent la détection en amont des défauts avant que les tests dynamiques ne soient effectués (par exemple, lors de revues des exigences ou des spécifications de conception, l'affinement du backlog du produit, etc.). Les défauts détectés tôt sont souvent beaucoup moins chers à corriger que les défauts trouvés plus tard dans le cycle de vie, surtout en comparaison aux défauts trouvés après le déploiement et l'utilisation opérationnelle du logiciel.

Il est en effet presque toujours beaucoup moins coûteux pour l'organisation d'utiliser des techniques de tests statiques pour trouver des défauts et les corriger rapidement que d'utiliser des tests dynamiques pour trouver des défauts dans l'objet de test et ensuite les corriger, surtout lorsque l'on considère les coûts supplémentaires associés à la mise à jour d'autres produits d'activités et à l'exécution de tests de confirmation et de régression.

Parmi les autres avantages des tests statiques, on peut citer les points suivants :

- Détection et correction plus efficace des défauts, avant l'exécution des tests dynamiques
- Identification des défauts qui ne sont pas facilement décelables par des tests dynamiques
- Prévention des défauts de conception ou de codage par la découverte d'incohérences, d'ambiguïtés, de contradictions, d'omissions, d'inexactitudes et de redondances dans les exigences
- Augmentation de la productivité du développement (par exemple, grâce à une meilleure conception, à un code plus facile à maintenir)
- Réduction des coûts et des délais de développement
- Réduction des coûts et des délais des tests
- Réduction du coût total de la qualité tout au long de la durée de vie du logiciel, grâce à la réduction du nombre de défaillances plus tard dans le cycle de vie ou après la mise en service
- Amélioration de la communication entre les membres de l'équipe au cours de la participation aux revues

3.1.3 Différences entre les tests statiques et dynamiques

Les tests statiques et dynamiques peuvent avoir les mêmes objectifs (voir section 1.1.1), tels que fournir une évaluation de la qualité des produits d'activités et identifier les défauts le plus tôt possible. Les tests statiques et dynamiques se complètent mutuellement en trouvant différents types de défauts.

L'une des principales distinctions est que les tests statiques détectent directement les défauts dans les produits d'activités plutôt que d'identifier les défaillances causées par des défauts lorsque le logiciel est exécuté. Un défaut peut résider dans un produit d'activités pendant très longtemps sans causer une défaillance. Le chemin où se trouve le défaut peut être rarement exercé ou difficile à atteindre, de sorte qu'il ne sera pas facile de construire et d'exécuter un test dynamique qui mette en évidence ce défaut. Les tests statiques peuvent permettre de trouver le défaut avec beaucoup moins d'efforts.

Une autre distinction est que les tests statiques peuvent être utilisés pour améliorer la cohérence et la qualité interne des produits d'activités, tandis que les tests dynamiques se concentrent généralement sur les comportements visibles de l'extérieur.

Par rapport aux tests dynamiques, les défauts qui sont habituellement plus faciles et moins coûteux à trouver et à corriger grâce aux tests statiques sont les suivants :

- Défauts dans les exigences (p. ex. incohérences, ambiguïtés, contradictions, omissions, inexactitudes et redondances)
- Défauts dans la conception (p. ex. algorithmes ou structures de base de données inefficaces, taux de dépendance élevé, faible cohésion)
- Défauts dans le code (p. ex. variables avec des valeurs non définies, variables déclarées mais jamais utilisées, code inatteignable, code dupliqué)

- Ecarts par rapport aux normes (p. ex. non-respect des règles de codage)
- Spécifications d'interface incorrectes (p. ex. unités de mesure utilisées par le système appelant différentes de celles utilisées par le système appelé)
- Vulnérabilités de sécurité (p. ex. sensibilité aux débordements de la mémoire tampon)
- Lacunes ou inexactitudes dans la traçabilité ou la couverture des bases de test (p. ex., des tests manquants pour un critère d'acceptation)

De plus, la plupart des défauts de maintenabilité ne peuvent être trouvés que par des tests statiques (par exemple, une modularité inadéquate, une mauvaise réutilisation des composants, un code difficile à analyser et à modifier sans introduire de nouveaux défauts).

3.2 Processus de revue

Les revues varient d'informelles à formelles. Les revues informelles se caractérisent par le fait qu'elles ne suivent pas un processus défini et n'ont pas de résultats formels documentés. Les revues formelles sont caractérisées par la participation de l'équipe, la documentation des résultats de la revue et la documentation des procédures pour la conduite de la revue Le caractère formel d'un processus de revue est lié à des facteurs tels que le modèle de cycle de vie de développement du logiciel, la maturité du processus de développement, la complexité du produit d'activités à revoir, toute exigence légale ou réglementaire, et/ou la nécessité d'une traçabilité d'audit.

La focalisation d'une revue dépend des objectifs fixés pour la revue (par exemple, trouver des défauts, gagner en compréhension, former les participants tels que des testeurs et de nouveaux membres de l'équipe, ou discuter et décider par consensus).

La norme ISO (ISO/CEI 20246) contient des descriptions plus détaillées du processus de revue des produits d'activités, y compris les rôles et les techniques de revue.

3.2.1 Processus de revue de produits d'activités

Le processus de revue comprend les principales activités suivantes :

Planification

- Définir le périmètre, qui comprend le but de la revue, les documents ou parties de documents à revoir et les caractéristiques de qualité à évaluer
- Estimer l'effort et le temps requis
- Identifier les caractéristiques de la revue telles que le type de revue avec les rôles, les activités et les checklists
- Sélectionner les personnes qui participeront à la revue et attribuer des rôles
- Définir des critères d'entrée et de sortie pour les types de revue plus formels (p. ex. inspections)
- Vérifier que les critères d'entrée soient satisfaits (pour les types de revue plus formels)

Lancement de la revue

- Distribuer le produit d'activités (physiquement ou par voie électronique) et d'autres documents, comme des formulaires de saisie des défauts, des checklists et des produits d'activités connexes
- Expliquer le périmètre, les objectifs, le processus, les rôles et les produits d'activités aux participants

- Répondre à toutes les questions que les participants peuvent avoir au sujet de la revue

Revue individuelle (c.-à-d. préparation individuelle)

- Revoir tout ou partie du produit d'activités
- Noter les défauts potentiels, les recommandations et les questions

Communication et analyse des problèmes

- Communiquer les défauts potentiels identifiés (p. ex. lors d'une réunion de revue)
- Analyser les défauts potentiels, leur affecter un responsable et un statut
- Évaluer et documenter les caractéristiques de qualité
- Évaluer les résultats de la revue en fonction des critères de sortie pour prendre une décision de revue (rejet ; changements majeurs nécessaires ; acceptation, éventuellement avec des changements mineurs)

Correction et production de rapports

- Produire des rapports de défauts pour les constatations qui nécessitent des changements
- Corriger les défauts trouvés (correction généralement réalisée par l'auteur) dans le produit d'activités revu
- Communiquer les défauts à la personne ou à l'équipe concernée (lorsqu'ils se trouvent dans un produit d'activités lié au produit d'activités revu)
- Enregistrer l'état actualisé des défauts (dans les revues formelles), y compris éventuellement l'accord de l'auteur du commentaire concerné
- Recueillir des métriques (pour les types de revue plus formels)
- Vérifier que les critères de sortie sont satisfaits (pour les types de revue plus formels)
- Accepter le produit d'activités lorsque les critères de sortie sont satisfaits

Les résultats de la revue d'un produit d'activités varient selon le type de revue et le degré de formalisme, tel que décrit à la section 3.2.3.

3.2.2 Rôles et responsabilités dans une revue formelle

Une revue formelle comprendra les rôles suivants :

Auteur

- Crée le produit d'activités revu
- Corrige les défauts du produit d'activités revu (si nécessaire)

Manager

- Est responsable de la planification de la revue
- Décide de la mise en œuvre des revues
- Affecte le personnel, le budget et le temps
- Vérifie le rapport coût-efficacité en continu

- Met en œuvre les mesures appropriées en cas de résultats inadéquats

Facilitateur (souvent appelé modérateur)

- Assure le bon déroulement des réunions de revue (quand elles ont lieu)
- Fait la médiation, si nécessaire, entre les différents points de vue
- Est souvent la personne dont dépend le succès de la revue

Responsable de la revue

- Prend la responsabilité générale de la revue
- Décide qui sera impliqué et organise quand et où elle aura lieu

Réviseurs

- Il peut s'agir d'experts du domaine, de personnes travaillant sur le projet, d'intervenants ayant un intérêt pour le produit d'activités et/ou de personnes ayant des compétences techniques ou métier spécifiques
- Ils identifient les défauts potentiels du produit d'activités à revoir
- Ils peuvent représenter différentes perspectives (p. ex. testeur, programmeur, utilisateur, opérateur, analyste métier, expert en utilisabilité, etc.).

Scribe (ou rapporteur)

- Recueille les défauts potentiels découverts au cours de l'activité de revue individuelle
- Enregistre les nouveaux défauts potentiels, les points en suspens et les décisions prises lors de la réunion de revue (durant son déroulement)

Dans certains types de revue, une personne peut endosser plusieurs rôles, et les actions associées à chaque rôle peuvent également varier en fonction du type de revue. En outre, avec des outils d'aide à la revue, en particulier la saisie des défauts, des points ouverts et des décisions, il est souvent inutile d'avoir recours à un scribe.

Des rôles plus détaillés sont possibles, comme décrit dans la norme ISO (ISO/CEI 20246).

3.2.3 Types de revue

Bien que les revues puissent être utilisées à diverses fins, l'un des principaux objectifs est de découvrir des défauts. Tous les types de revue peuvent aider à la détection des défauts, et le type de revue sélectionné doit être basé sur les besoins du projet, les ressources disponibles, le type de produit et les risques, le domaine d'activité et la culture de l'entreprise, entre autres critères de sélection.

Les revues peuvent être classées en fonction de leurs différentes caractéristiques. Voici la liste des quatre types de revues les plus courants et les caractéristiques qui leur sont associées.

Revue informelle (p. ex., par un collègue, en binôme, revue de pair)

- Objectif principal : détecter d'éventuels défauts
- Autres objectifs possibles : générer de nouvelles idées ou solutions, résoudre rapidement des problèmes mineurs
- Ne repose pas sur un processus formel (documenté)
- Peut ne pas comporter de réunion de revue

- Peut être effectué par un collègue de l'auteur ou par d'autres personnes
- Les résultats peuvent être documentés
- L'utilité varie selon les réviseurs
- L'utilisation de checklists est facultative
- Très couramment utilisé dans le développement Agile

Relecture technique

- Principaux objectifs : trouver des défauts, améliorer le produit logiciel, envisager des implémentations alternatives, évaluer la conformité aux normes et aux spécifications
- Autres objectifs possibles : échange d'idées sur des techniques ou des variations de style, formation des participants, obtention d'un consensus
- La préparation individuelle avant la réunion de revue est facultative
- La réunion de revue est généralement dirigée par l'auteur du produit d'activités
- Le rôle de scribe est obligatoire
- L'utilisation de checklists est facultative
- Peut prendre la forme de scénarios, d'essais à blanc ou de simulations
- Des rapports de défauts et des rapports de revue peuvent être produits
- Peut varier dans la pratique de plutôt informel à très formel

Revue technique

- Principaux objectifs : obtention d'un consensus, détection de défauts potentiels
- Autres objectifs possibles : évaluer la qualité et renforcer la confiance dans le produit d'activités revu, générer de nouvelles idées, motiver et permettre aux auteurs d'améliorer les produits d'activités futurs, envisager des implémentations alternatives
- Les réviseurs doivent être des pairs techniques de l'auteur et des experts techniques dans la même discipline ou dans d'autres disciplines
- Une préparation individuelle avant la réunion de revue est requise
- La réunion de revue est facultative, et de préférence dirigée par un facilitateur formé (généralement pas l'auteur)
- Le rôle de scribe est obligatoire, de préférence pas l'auteur
- L'utilisation de checklists est facultative
- Des rapports de défauts et des rapports de revue sont généralement produits

Inspection

- Principaux objectifs : détecter les défauts potentiels, évaluer la qualité et renforcer la confiance dans le produit d'activités, prévenir de futurs défauts similaires par l'apprentissage de l'auteur et l'analyse des causes racines
- Autres objectifs possibles : motiver et permettre aux auteurs d'améliorer les futurs produits d'activités et le processus de développement de logiciels, parvenir à un consensus

- Suit un processus défini avec des résultats formels documentés, basé sur des règles et des checklists
- Utilise des rôles clairement définis, comme ceux spécifiés à la section 3.2.2 qui sont obligatoires, et peut inclure un lecteur spécialisé (qui lit le produit d'activités à haute voix pendant la réunion de revue)
- Une préparation individuelle avant la réunion de revue est requise
- Les réviseurs sont soit des pairs de l'auteur, soit des experts dans d'autres disciplines pertinentes pour le produit d'activités concerné
- Des critères d'entrée et de sortie spécifiés sont utilisés
- Le rôle de scribe est obligatoire
- La réunion de revue est dirigée par un facilitateur formé (pas l'auteur)
- L'auteur ne peut pas agir à titre de responsable de la revue, de lecteur ou de scribe
- Des rapports de défauts et des rapports de revue sont produits
- Des métriques sont collectées et utilisées pour améliorer l'ensemble du processus de développement logiciel, y compris le processus d'inspection

Un même produit d'activités peut faire l'objet de plus d'un type de revue. Si plus d'un type de revue est utilisé, l'ordre peut varier. Par exemple, une revue informelle peut être effectuée avant une revue technique, afin de s'assurer que le produit d'activités est prêt pour une revue technique.

Les types de revue décrits ci-dessus peuvent être effectués sous forme de revue par les pairs, c'est-à-dire par des collègues à un niveau organisationnel sensiblement similaire.

Les types de défauts constatés lors d'une revue varient principalement en fonction du produit d'activités examiné. Voir la section 3.1.3 pour des exemples de défauts qui peuvent être trouvés par des revues dans différents produits d'activités, et voir Gilb 1993 pour des informations sur les inspections formelles.

3.2.4 Application des techniques de revue

Il existe un ensemble de techniques de revue qui peuvent être appliquées au cours de l'activité de revue individuelle (c.-à-d., préparation individuelle) pour découvrir des défauts. Ces techniques peuvent être utilisées pour tous les types de revue décrits ci-dessus. L'efficacité des techniques peut varier selon le type de revue utilisée. Voici des exemples de différentes techniques de revue individuelles pour différents types de revue.

Ad hoc

Dans le cadre d'une revue ad hoc, les réviseurs reçoivent peu ou pas de directives sur la façon dont cette tâche devrait être accomplie. Les réviseurs examinent généralement le produit d'activités de façon séquentielle, en identifiant et en documentant les problèmes au fur et à mesure qu'ils les rencontrent. La révision ad hoc est une technique couramment utilisée qui nécessite peu de préparation. Cette technique dépend fortement des compétences des réviseurs et peut donner lieu à des constats qui font double emploi car signalés par différents réviseurs.

Basée sur les checklists

Une revue basée sur une checklist est une technique systématique, pour laquelle les réviseurs détectent les problèmes sur la base de checklist qui sont distribuées au début de la revue (par exemple, par le facilitateur). Une checklist consiste en une série de questions basées sur des défauts potentiels, qui

peuvent être issus de l'expérience. Les checklists devraient être spécifiques au type de produit d'activités examiné et devraient être mises à jour régulièrement pour couvrir les types de problèmes qui ont été omis lors de revues précédentes. Le principal avantage de la technique basée sur les checklists est la couverture systématique des types de défauts courants. Il faut prendre soin de ne pas simplement suivre la checklist lors de la revue individuelle, mais aussi de rechercher des défauts en dehors de la checklist.

Scénarios et essais à blanc

Dans le cadre d'une revue fondée sur des scénarios, les réviseurs reçoivent un guide structuré sur la façon de lire le produit d'activités. Une approche fondée sur des scénarios aide les réviseurs à effectuer des "essais à blanc" sur le produit d'activités en fonction de l'utilisation prévue du produit d'activités (si le produit d'activités est documenté dans un format approprié, comme les cas d'utilisation par exemple). Ces scénarios permettent aux réviseurs d'être mieux guidés sur la façon d'identifier des types de défauts spécifiques qu'avec uniquement les éléments d'une checklist. Comme pour les revues basées sur des checklists, afin de ne pas manquer certains types de défauts (p. ex., des caractéristiques manquantes), les réviseurs ne devraient pas être limités aux scénarios documentés.

Basée sur les rôles

Une revue basée sur les rôles est une technique par laquelle les réviseurs évaluent le produit d'activités du point de vue de rôles individuels des parties prenantes. Les rôles typiques comprennent des types d'utilisateurs finaux spécifiques (expérimentés, inexpérimentés, seniors, enfants, etc.) et des rôles spécifiques dans l'organisation (administrateur utilisateur, administrateur système, testeur de performance, etc.).

Basée sur la perspective

Dans une lecture basée sur la perspective, de façon semblable à une revue basée sur les rôles, les réviseurs adoptent différents points de vue des parties prenantes dans le cadre d'une revue individuelle. Les points de vue habituels des parties prenantes comprennent l'utilisateur final, le marketing, le concepteur, le testeur ou les opérations. L'utilisation de différents points de vue des parties prenantes permet d'approfondir la revue individuelle avec moins de doublons de défauts entre les réviseurs.

De plus, la lecture basée sur la perspective exige également que les réviseurs essaient d'utiliser le produit d'activités revu pour produire le produit qu'ils en tireraient. Par exemple, un testeur pourrait essayer de générer des brouillons de tests d'acceptation en réalisant une lecture basée sur la perspective d'une spécification des exigences pour voir si toute l'information nécessaire est bien incluse. De plus, dans le cadre de la lecture basée sur la perspective, on s'attend à ce que des checklists soient utilisées.

Des études empiriques ont montré que la lecture basée sur la perspective est la technique générale la plus efficace pour revoir des exigences et des produits d'activités techniques. Un facteur clé de succès est l'inclusion et la prise en compte appropriée de différents points de vue des parties prenantes, en fonction des risques. Voir Shul 2000 pour plus de détails sur la lecture basée sur la perspective, et Sauer 2000 pour l'efficacité des différents types de revue.

3.2.5 Facteurs de réussite des revues

Pour que la revue soit réussie, il faut tenir compte du type de revue approprié et des techniques utilisées. De plus, un certain nombre d'autres facteurs influent sur les résultats de la revue.

Les facteurs de réussite organisationnels pour les revues incluent :

- Chaque revue a des objectifs clairs, définis lors de la planification de la revue et utilisés comme critères de sortie mesurables

- Le type de revue est choisi en fonction des objectifs, du type et du niveau des produits d'activités logiciels, mais aussi en tenant compte des participants
- Toutes les techniques de revue utilisées, telles que la revue basée sur les checklists ou sur les rôles, sont appropriées pour l'identification efficace des défauts dans le produit d'activités à revoir
- Toutes les checklists utilisées couvrent les principaux risques et sont actualisées
- Les documents volumineux sont rédigés et révisés en petits morceaux, de sorte que le contrôle de la qualité est réalisé en fournissant aux auteurs un feed-back précoce et fréquent sur les défauts
- Les participants ont suffisamment de temps pour préparer
- Les revues sont planifiées avec un délai de notification suffisant
- Le management appuie le processus de revue (p. ex. en intégrant suffisamment de temps pour les activités de revue dans les échéanciers des projets)

Les facteurs de succès liés aux participants aux revues sont les suivants :

- Le choix des bonnes personnes contribue à l'atteinte des objectifs de la revue, par exemple, des personnes ayant des compétences ou des perspectives différentes, qui peuvent utiliser le document comme intrant de leurs activités
- Les testeurs sont considérés comme des réviseurs appréciés qui contribuent à la revue et aussi apprennent à propos du produit d'activités revu, ce qui leur permet de produire des tests plus efficaces et de préparer ces tests plus tôt
- Les participants consacrent suffisamment de temps et d'attention aux détails
- Les revues sont effectuées sur de petits morceaux, de sorte que les réviseurs ne perdent pas leur concentration pendant la revue individuelle et/ou la réunion de revue (quand elle a lieu)
- Les défauts trouvés sont identifiés, compris et traités avec objectivité
- La réunion est bien gérée, de sorte que les participants considèrent qu'il s'agit d'une utilisation efficace de leur temps
- La revue est menée dans un climat de confiance ; le résultat ne sera pas utilisé pour l'évaluation des participants
- Les participants évitent le langage corporel et les comportements qui pourraient indiquer l'ennui, l'exaspération ou l'hostilité envers les autres participants
- Une formation adéquate est fournie, en particulier pour les types de revue plus formels tels que les inspections
- Une culture de l'apprentissage et de l'amélioration des processus est encouragée

Voir Gilb 1993, Wiegers 2002, et van Veenendaal 2004 pour plus d'informations sur des revues efficaces.

4 Techniques de test — 330 minutes

Termes

technique de test de boîte-noire, analyse des valeurs limites, test basé sur les checklists, couverture, couverture des décisions, test de tables de décision, estimation d'erreur, partitions d'équivalence, technique de test basée sur l'expérience, test exploratoire, test des transitions d'état, couverture des instructions, technique de test, test des cas d'utilisation, technique de test de boîte-blanche

Objectifs d'apprentissage pour les techniques de test

4.1 Catégories de techniques de test

FL-4.1.1 (K2) Expliquer les caractéristiques, les points communs et différences entre les techniques de test boîte-noire, boîte-blanche et basées sur l'expérience

4.2 Techniques de test boîte-noire

FL-4.2.1 (K3) Appliquer la technique des partitions d'équivalence pour produire des cas de test à partir d'exigences données

FL-4.2.2 (K3) Appliquer l'analyse des valeurs limites pour produire des cas de test à partir d'exigences données

FL-4.2.3 (K3) Appliquer le test de tables de décision pour produire des cas de test à partir d'exigences données

FL-4.2.4 (K3) Appliquer le test des transitions d'état pour produire des cas de test à partir d'exigences données

FL-4.2.5 (K2) Expliquer comment produire des cas de test à partir d'un cas d'utilisation

4.3 Techniques de test boîte-blanche

FL-4.3.1 (K2) Expliquer la couverture des instructions

FL-4.3.2 (K2) Expliquer la couverture des décisions

FL-4.3.3 (K2) Expliquer l'intérêt de la couverture des instructions et des décisions

4.4 Techniques de test basées sur l'expérience

FL-4.4.1 (K2) Expliquer l'estimation d'erreur

FL-4.4.2 (K2) Expliquer le test exploratoire

FL-4.4.3 (K2) Expliquer le test basé sur des checklists

4.1 Catégories de techniques de test

L'objectif d'une technique de test, y compris celles présentées dans cette section, est d'aider à identifier les conditions de test, les cas de test et les données de test.

4.1.1 Choix des techniques de test

Le choix des techniques de test à utiliser dépend d'un certain nombre de facteurs, dont les suivants :

- Type de composant ou de système
- Complexité du composant ou des systèmes
- Normes réglementaires
- Exigences client ou contractuelles
- Niveaux de risque
- Types de risques
- Objectifs du test
- Documentation disponible
- Connaissances et compétences des testeurs
- Outils disponibles
- Temps et budget
- Modèle de cycle de vie du développement logiciel
- Utilisation prévue du logiciel
- Expérience antérieure de l'utilisation des techniques de test sur le composant ou le système à tester
- Types de défauts attendus dans le composant ou le système

Certaines techniques sont plus applicables à certaines situations et à certains niveaux de test ; d'autres sont applicables à tous les niveaux de test. Lors de la création de cas de test, les testeurs utilisent généralement une combinaison de techniques de test pour obtenir les meilleurs résultats de l'effort de test.

L'utilisation des techniques de test dans les activités d'analyse des tests, de conception des tests et d'implémentation des tests peut varier de très informel (peu ou pas de documentation) à très formel. Le niveau de formalité approprié dépend du contexte des tests, y compris la maturité des processus de test et de développement, les contraintes de temps, les exigences de sûreté ou réglementaires, les connaissances et les compétences des personnes impliquées et le modèle de cycle de vie du développement logiciel suivi.

4.1.2 Catégories de techniques de test et leurs caractéristiques

Dans ce syllabus, les techniques de test sont classées en techniques boîte-noire, techniques boîte-blanche et techniques basées sur l'expérience.

Les techniques de test de boîte-noire (aussi appelées techniques comportementales ou techniques basées sur le comportement) sont basées sur une analyse de la base de test appropriée (par exemple, documents d'exigences formelles, spécifications, cas d'utilisation, User Stories ou processus métier). Ces techniques sont applicables aux tests fonctionnels et non-fonctionnels. Les techniques de test de boîte-noire se concentrent sur les entrées et sorties de l'objet de test sans référence à sa structure interne.

Les techniques de test de boite blanche (aussi appelées techniques structurelles ou techniques basées sur la structure) sont basées sur une analyse de l'architecture, de la conception détaillée, de la structure interne ou du code de l'objet de test. Contrairement aux techniques de test de boîte-noire, les techniques de test de boîte-blanche se concentrent sur la structure et le traitement à l'intérieur de l'objet de test.

Les techniques de test basées sur l'expérience tirent parti de l'expérience des développeurs, des testeurs et des utilisateurs pour concevoir, implémenter et exécuter des tests. Ces techniques sont souvent combinées à des techniques de test boîte-noire et boite blanche.

Les caractéristiques communes des techniques de test boîte-noire incluent :

- Les conditions de test, les cas de test et les données de test sont dérivés d'une base de test qui peut inclure des exigences logicielles, des spécifications, des cas d'utilisation, et des User Stories
- Les cas de test peuvent être utilisés pour détecter les écarts entre les exigences et l'implémentation des exigences, ainsi que les défauts au niveau des exigences
- La couverture est mesurée en fonction des éléments de la base de test évalués et de la technique appliquée à cette base de test

Les caractéristiques communes des techniques de test des boîte-blanche incluent :

- Les conditions de test, les cas de test et les données de test sont dérivés d'une base de test qui peut inclure le code, l'architecture logicielle, la conception détaillée ou toute autre source d'information concernant la structure du logiciel
- La couverture est mesurée en fonction des éléments testés au sein d'une structure donnée (p. ex. le code ou les interfaces)
- Les spécifications sont souvent utilisées comme source d'information supplémentaire pour déterminer le résultat attendu des cas de test

Les caractéristiques communes des techniques de test basées sur l'expérience incluent :

- Les conditions de test, les cas de test et les données de test sont dérivés d'une base de test qui peut inclure les connaissances et l'expérience des testeurs, développeurs, utilisateurs et autres parties prenantes

Ces connaissances et cette expérience comprennent l'utilisation prévue du logiciel, son environnement, les défauts probables et la répartition de ces défauts.

La norme internationale (ISO/IEC/IEEE 29119-4) contient des descriptions des techniques de test et les mesures de couverture correspondantes (voir Craig 2002 et Copeland 2004 pour plus d'informations sur ces techniques).

4.2 Techniques de test boîte-noire

4.2.1 Partitions d'équivalence

Les partitions d'équivalence divisent les données en partitions (également connues sous le nom de classes d'équivalence) de telle sorte que tous les éléments d'une partition donnée sont supposés être traités de la même manière (voir Kaner 2013 et Jorgensen 2014). Il y a des partitions d'équivalence pour les valeurs valides et invalides.

- Les valeurs valides sont des valeurs qui doivent être acceptées par le composant ou le système. Une partition d'équivalence contenant des valeurs valides est appelée "partition d'équivalence valide".
- Les valeurs invalides sont des valeurs qui doivent être rejetées par le composant ou le système. Une partition d'équivalence contenant des valeurs invalides est appelée "partition d'équivalence invalide".
- Les partitions peuvent être identifiées pour tout élément de données lié à l'objet de test, y compris les entrées, les sorties, les valeurs internes, les valeurs liées au temps (par exemple, avant ou après un événement) et pour les paramètres d'interface (par exemple, les composants intégrés testés pendant les tests d'intégration).
- Toute partition peut être divisée en sous-partitions si nécessaire.
- Chaque valeur doit appartenir à une et une seule partition d'équivalence.
- Lorsque des partitions d'équivalence invalides sont utilisées dans des cas de test, elles doivent être testées individuellement, c'est-à-dire qu'elles ne doivent pas être combinées avec d'autres partitions d'équivalence invalides, afin de s'assurer que les défaillances ne sont pas masquées. Les défaillances peuvent être masquées lorsque plusieurs défaillances surviennent en même temps, mais qu'une seule est visible, ce qui fait que les autres défaillances ne sont pas détectées.

Pour obtenir une couverture de 100% avec cette technique, les cas de test doivent couvrir toutes les partitions identifiées (y compris les partitions invalides) en utilisant au moins une valeur de chaque partition. La couverture est mesurée comme étant le nombre de partitions d'équivalence testées par au moins une valeur, divisé par le nombre total de partitions d'équivalence identifiées, généralement exprimé en pourcentage. Les partitions d'équivalence sont applicables à tous les niveaux de test.

4.2.2 Analyse des valeurs limites

L'analyse des valeurs limites est une extension des partitions d'équivalence, mais ne peut être utilisée que lorsque la partition est ordonnée, composée de données numériques ou séquentielles. Les valeurs minimale et maximale (ou première et dernière valeurs) d'une partition sont ses valeurs limites (Beizer 1990).

Par exemple, supposons qu'un champ d'entrée accepte une seule valeur entière comme entrée, en utilisant un clavier pour limiter les entrées de sorte que les entrées non entières sont impossibles. La plage valide va de 1 à 5 inclus. Il y a donc trois partitions d'équivalence : invalide (trop bas) ; valide ; invalide (trop élevé). Pour la partition d'équivalence valide, les valeurs limites sont 1 et 5. Pour la partition invalide (trop élevée), les valeurs limites sont 6 et 9. Pour la partition invalide (trop peu élevée), il n'y a qu'une seule valeur limite, 0, car il s'agit d'une partition à un seul élément.

Dans l'exemple ci-dessus, nous identifions deux valeurs limites par limite. La limite entre invalide (trop bas) et valide donne les valeurs de test 0 et 1 ; la limite entre valide et invalide (trop élevé) donne les valeurs de test 5 et 6. Certaines variantes de cette technique identifient trois valeurs limites par limite : les valeurs en-dessous, sur et juste au-dessus de la limite. Dans l'exemple précédent, en utilisant 3 valeurs par limite, les valeurs limites inférieures sont 0, 1 et 2, et les valeurs limites supérieures sont 4, 5, et 6 (Jorgensen 2014).

Le comportement aux limites des partitions d'équivalence est plus susceptible d'être incorrect que le comportement à l'intérieur des partitions. Il est important de se rappeler que les limites spécifiées et implémentées peuvent être décalées vers des positions supérieures ou inférieures à leurs valeurs prévues, être oubliées où être complétées avec des limites supplémentaires non souhaitées. L'analyse et le test des valeurs limites permettront de détecter presque tous ces défauts en forçant le logiciel à exhiber des comportements pour une partition autre que celle à laquelle la valeur limite devrait appartenir.

L'analyse des valeurs limites peut être appliquée à tous les niveaux de test. Cette technique est généralement utilisée pour tester les exigences qui nécessitent une série de valeurs numériques (y compris les dates et les heures). La couverture des limites d'une partition est mesurée comme le nombre de valeurs limites testées, divisé par le nombre total de valeurs limites identifiées, généralement exprimé en pourcentage.

4.2.3 Test de tables de décision

Les techniques de test combinatoire sont utiles pour tester la mise en œuvre des exigences du système qui spécifient comment différentes combinaisons de conditions donnent des résultats différents. L'une de ces techniques de test est le test de tables de décision.

Les tables de décision sont un bon moyen pour répertorier les règles métier complexes qu'un système doit mettre en œuvre. Lors de la création de tables de décision, le testeur identifie les conditions (souvent des entrées) et les actions résultantes (souvent des sorties) du système. Celles-ci forment les lignes du tableau, généralement avec les conditions en haut et les actions en bas. Chaque colonne correspond à une règle de décision qui définit une combinaison unique de conditions qui aboutit à l'exécution des actions associées à cette règle. Les valeurs des conditions et des actions sont généralement affichées sous forme de valeurs booléennes (vraies ou fausses) ou de valeurs discrètes (par exemple, rouge, vert, bleu), mais peuvent également être des nombres ou des plages de nombres. Ces différents types de conditions et d'actions peuvent être regroupés dans la même table.

La notation courante dans les tables de décision est la suivante :

Pour les conditions :

- V signifie que la condition est vraie (peut aussi être représentée par O (Oui) ou 1)
- F signifie que la condition est fausse (peut aussi être affiché comme N (Non) ou 0)
- — signifie que la valeur de la condition n'a pas d'importance (peut également être affichée en tant que N/A)

Pour les actions :

- X signifie que l'action doit avoir lieu (peut aussi être représenté par V, O ou 1)
- Vide signifie que l'action ne doit pas se produire (peut également être affiché sous la forme - ou F, N ou 0)

Une table de décision complète comporte suffisamment de colonnes pour couvrir toutes les combinaisons de conditions. La table peut être réduite en supprimant les colonnes contenant des combinaisons impossibles de conditions, les colonnes contenant des combinaisons de conditions possibles mais irréalisables et les colonnes qui testent des combinaisons de conditions qui n'affectent pas le résultat. Pour plus d'informations sur la façon de réduire les tables de décision, se référer au syllabus CFTL / ISTQB Testeur Certifié de Niveau Avancé Analyste de Test.

La couverture minimale courante pour le test de tables de décision est d'avoir au moins un cas type par règle de décision dans la table. Il s'agit généralement de couvrir toutes les combinaisons de conditions. La couverture est mesurée comme le nombre de règles de décision testées par au moins un cas de test, divisé par le nombre total de règles de décision, généralement exprimé en pourcentage.

La force de la technique de test utilisant les tables de décision est qu'elle permet d'identifier toutes les combinaisons importantes de conditions, dont certaines pourraient sinon être négligées. Cette technique permet également de trouver des lacunes dans les exigences. Elle peut être appliquée à toutes les situations dans lesquelles le comportement du logiciel dépend d'une combinaison de conditions, à n'importe quel niveau de test.

4.2.4 Test des transitions d'état

Les composants ou les systèmes peuvent réagir différemment à un événement en fonction des conditions présentes ou de leur historique (par exemple, les événements qui se sont produits depuis l'initialisation du système). L'historique antérieur peut être résumé à l'aide du concept d'états. Un diagramme de transitions d'état montre les états possibles du logiciel, ainsi que la façon dont le logiciel entre, sort et évolue entre ces états. Une transition est déclenchée par un événement (par exemple, l'entrée d'une valeur par l'utilisateur dans un champ). L'événement entraîne une transition. Si le même événement peut entraîner deux transitions différentes ou plus à partir d'un même état, cet événement peut être qualifié par une condition de garde. Le changement d'état peut entraîner une action du logiciel (par exemple, l'affichage du résultat d'un calcul ou d'un message d'erreur).

Un tableau de transition d'état montre toutes les transitions valides et les transitions potentiellement invalides entre les états, ainsi que les événements, les conditions de garde et les actions résultantes pour les transitions valides. Les diagrammes de transition d'états ne montrent normalement que les transitions valides et excluent les transitions invalides.

Les tests peuvent être conçus pour couvrir une séquence de plusieurs états, pour exercer tous les états, pour exercer toutes les transitions, pour exercer des séquences spécifiques de transitions ou pour tester des transitions invalides.

Le test de transitions d'état est utilisé pour les applications basées sur des menus et est aussi couramment utilisé dans l'industrie du logiciel embarqué. La technique convient également à la modélisation d'un scénario métier ayant des états spécifiques ou pour tester la navigation à l'écran. Le concept d'état est abstrait, il peut représenter quelques lignes de code ou un processus métier entier.

La couverture est généralement mesurée comme étant le nombre d'états ou de transitions identifiés testés, divisé par le nombre total d'états ou de transitions identifiés dans l'objet de test, normalement exprimé en pourcentage. Pour plus d'informations sur les critères de couverture pour le test des transitions d'état, se référer au syllabus CFTL / ISTQB Testeur Certifié de Niveau Avancé Analyste de Test.

4.2.5 Test des cas d'utilisation

Les tests peuvent être dérivés de cas d'utilisation. Les cas d'utilisation sont une façon spécifique de concevoir les interactions avec le logiciel pour représenter des exigences. Les cas d'utilisation sont

associés à des acteurs (utilisateurs humains, matériel externe ou autres composants ou systèmes) et à des sujets (le composant ou système auquel le cas d'utilisation est appliqué).

Chaque cas d'utilisation spécifie un comportement qu'un sujet peut accomplir en collaboration avec un ou plusieurs acteurs (UML 2.5.1 2017). Un cas d'utilisation peut être décrit par des interactions et des activités, ainsi que par des préconditions, des post conditions et du langage naturel. Les interactions entre les acteurs et le sujet peuvent entraîner des changements dans l'état du sujet. Les interactions peuvent être représentées graphiquement par des flux de travail, des diagrammes d'activités ou des modèles de processus métier.

Un cas d'utilisation peut inclure des variations possibles de son comportement de base, y compris un comportement exceptionnel et la gestion des erreurs (réponse du système et récupération à la suite d'erreurs de programmation, au niveau de l'application et de communication, par exemple, entraînant un message d'erreur). Les tests sont conçus pour exercer les comportements définis (comportement de base, exceptionnel ou alternatif, et traitement des erreurs). La couverture peut être mesurée par le pourcentage des comportements de cas d'utilisation testés divisé par le nombre total des comportements du cas d'utilisation, généralement exprimé en pourcentage.

Pour plus d'informations sur les critères de couverture pour le test des cas d'utilisation, se référer au syllabus CFTL / ISTQB Testeur Certifié de Niveau Avancé Analyste de Test.

4.3 Techniques de test boîte-blanche

Les tests boîte-blanche sont basés sur la structure interne de l'objet de test. Les techniques de test boîte-blanche peuvent être utilisées à tous les niveaux de test, mais les deux techniques liées au code dont il est question dans cette section sont le plus souvent utilisées au niveau du test des composants. Il existe des techniques plus avancées qui sont utilisées dans des contextes critiques pour la sûreté, critiques pour la mission ou à fort besoin de fiabilité, et visant à obtenir une couverture plus complète, mais ces techniques ne sont pas abordées ici. Pour plus d'informations sur ces techniques, se référer au syllabus CFTL / ISTQB Testeur Certifié de Niveau Avancé Analyste Technique de Test.

4.3.1 Test et couverture des instructions

Le test des instructions exerce les instructions exécutables dans le code. La couverture est mesurée comme le nombre d'instructions exécutées par les tests, divisé par le nombre total d'instructions exécutables dans l'objet de test, généralement exprimé en pourcentage.

4.3.2 Test et couverture des décisions

Le test des décisions exerce les décisions dans le code et teste le code qui est exécuté sur la base des résultats des décisions. Pour cela, les cas de test suivent les flux de contrôle qui se produisent à partir d'un point de décision (par exemple, pour une instruction IF, un pour le résultat vrai et un pour le résultat faux ; pour une instruction CASE, des cas de test seraient nécessaires pour tous les résultats possibles, y compris le résultat par défaut).

La couverture est mesurée comme le nombre de résultats de décision exécutés par les tests, divisé par le nombre total de résultats de décision dans l'objet de test, généralement exprimé en pourcentage.

4.3.3 Apport des tests des instructions et décisions

Lorsque la couverture à 100% des instructions est atteinte, elle garantit que toutes les instructions exécutables du code ont été testées au moins une fois, mais elle ne garantit pas que toutes les décisions

ont été testées. Des deux techniques de test boîte-blanche présentées dans ce syllabus, le test des instructions peut fournir une couverture du code moindre que le test des décisions.

Lorsque la couverture à 100 % des décisions est atteinte, tous les résultats des décisions sont exécutés, ce qui comprend le test du résultat vrai et aussi du résultat faux, même lorsqu'il n'y a pas d'instruction FALSE explicite. La couverture des instructions aide à trouver des défauts dans les parties du code qui n'ont pas été exercées par d'autres tests. La couverture des décisions aide à trouver des défauts dans le code lorsque d'autres tests n'ont pas couvert à la fois les résultats vrais et les résultats faux des décisions.

L'obtention d'une couverture à 100 % des décisions garantit une couverture à 100 % des instructions (mais pas l'inverse).

4.4 Techniques de test basées sur l'expérience

Lors de l'application de techniques de test basées sur l'expérience, les cas de test sont basés sur les compétences et l'intuition du testeur, ainsi que sur son expérience avec des applications et des technologies similaires. Ces techniques peuvent être utiles pour identifier les tests qui n'ont pas été facilement identifiés par d'autres techniques plus systématiques. Selon l'approche et l'expérience du testeur, ces techniques peuvent atteindre des degrés de couverture et d'efficacité très variables. La couverture peut être difficile à évaluer et peut ne pas être mesurable avec ces techniques.

Les techniques basées sur l'expérience les plus couramment utilisées sont abordées dans les sections suivantes.

4.4.1 Estimation d'erreur

L'estimation d'erreur est une technique utilisée pour anticiper les erreurs, les défauts et les défaillances, sur la base des connaissances du testeur, y compris :

- Comment l'application a fonctionné antérieurement
- Quels types d'erreurs les développeurs ont tendance à faire
- Les défaillances qui se sont produites dans d'autres applications

Une approche méthodique de la technique par estimation d'erreur consiste à créer une liste d'erreurs, de défauts et de défaillances possibles, et à concevoir des tests qui exposeront ces défaillances et les défauts qui les ont causées. Ces listes d'erreurs, de défauts et de défaillances peuvent être construites sur la base de l'expérience, de données sur les défauts et les défaillances, ou à partir de connaissances générales sur les causes des défaillances logicielles.

4.4.2 Tests exploratoires

Dans les tests exploratoires, des tests informels (non prédéfinis) sont conçus, exécutés, enregistrés et évalués dynamiquement pendant l'exécution des tests. Les résultats des tests sont utilisés pour en apprendre davantage sur le composant ou le système, et pour créer des tests pour les parties qui pourraient avoir besoin de plus de tests.

Les tests exploratoires sont parfois réalisés en utilisant des sessions de test pour structurer l'activité. Lors du test basé sur des sessions, les tests exploratoires sont effectués dans un temps défini, et le testeur utilise une charte de test comprenant les objectifs du test pour guider les tests. Le testeur peut utiliser des fiches de session de test pour documenter les étapes suivies et les constatations réalisées.

Les tests exploratoires sont le plus utile lorsqu'il y a peu de spécifications ou des spécifications inadéquates ou des contraintes de temps importantes sur les tests. Les tests exploratoires sont également utiles pour compléter d'autres techniques de test plus formelles.

Les tests exploratoires sont étroitement associés aux stratégies de test réactives (voir section 5.2.2). Les tests exploratoires peuvent inclure l'utilisation d'autres techniques boîte-noire, boîte-blanche et basées sur l'expérience.

4.4.3 Tests basés sur des checklists

Dans les tests basés sur des checklists, les testeurs conçoivent, implémentent et exécutent des tests pour couvrir les conditions de test figurant dans une checklist. Au cours de l'analyse, les testeurs créent une nouvelle checklist ou complètent une checklist existante, mais les testeurs peuvent également utiliser une checklist existante sans modification. De telles checklists peuvent être construites sur la base de l'expérience, de la connaissance de ce qui est important pour l'utilisateur, ou de la compréhension du pourquoi et du comment des défaillances logicielles.

Des checklists peuvent être créées pour prendre en charge différents types de tests, y compris les tests fonctionnels et non-fonctionnels. En l'absence de cas de tests détaillés, les tests basés sur des checklists peuvent fournir des lignes directrices et un certain degré de cohérence. Comme il s'agit de listes de haut niveau, il est probable qu'il y ait une certaine variabilité dans les tests réels, ce qui pourrait entraîner une plus grande couverture des tests, mais une reproductibilité plus faible.

5 Gestion des tests — 225 minutes

Termes

gestion de configuration, gestion des défauts, critères d'entrée, critères de sortie, risque produit, risque projet, risque, niveau de risque, test basé sur les risques, approche de test, contrôle des tests, estimation des tests, Test Manager, pilotage des tests, plan de test, planification des tests, rapport d'avancement des tests, stratégie de test, rapport de synthèse des tests, testeur

Objectifs d'apprentissage pour la gestion des tests

5.1 Organisation des tests

FL-5.1.1 (K2) Expliquer les bénéfices et inconvénients du test indépendant

FL-5.1.2 (K1) Identifier les tâches d'un Test Manager et d'un testeur

5.2 Planification et estimation des tests

FL-5.2.1 (K2) Résumer l'objectif et le contenu d'un plan de test

FL-5.2.2 (K2) Expliquer les différences entre plusieurs stratégies de test

FL-5.2.3 (K2) Donner des exemples de critères d'entrée et de critères de sortie possibles

FL-5.2.4 (K3) Appliquer les informations de priorité et de dépendances techniques et logiques pour ordonnancer l'exécution d'un ensemble donné de cas de test

FL-5.2.5 (K1) Identifier des facteurs qui influencent l'effort relatif au test

FL-5.2.6 (K2) Expliquer les différences entre deux techniques d'estimation : la technique basée sur des métriques et la technique basée sur l'expertise

5.3 Pilotage et contrôle des tests

FL-5.3.1 (K1) Se souvenir des métriques utilisées pour les tests

FL-5.3.2 (K2) Résumer les objectifs, contenus et destinataires des rapports de test

5.4 Gestion de configuration

FL-5.4.1 (K2) Résumer comment la gestion de configuration vient en appui des tests

5.5 Risques et tests

FL-5.5.1 (K1) Définir le niveau de risque à partir de la probabilité d'occurrence et de l'impact

FL-5.5.2 (K2) Décrire la différence entre les risques projet et les risques produit

FL-5.5.3 (K2) Décrire, en utilisant des exemples, comment l'analyse des risques produit peut influencer la complétude et le périmètre des tests

5.6 Gestion des défauts

FL-5.6.1 (K3) Ecrire un rapport de défaut couvrant les défauts trouvés pendant les tests

5.1 Organisation des tests

5.1.1 Indépendance des tests

Les tâches de test peuvent être réalisées par des personnes ayant un rôle spécifique aux tests, ou par des personnes ayant un autre rôle (par exemple, des clients). Un certain degré d'indépendance rend souvent le testeur plus efficace pour détecter les défauts du fait des différences entre les biais cognitifs de l'auteur et du testeur (voir section 1.5). L'indépendance ne remplace cependant pas la connaissance, et les développeurs peuvent aussi trouver efficacement de nombreux défauts dans leur propre code.

Les degrés d'indépendance dans les tests comprennent les niveaux suivants (allant d'un niveau d'indépendance faible à un niveau élevé) :

- Pas de testeurs indépendants ; la seule forme de test existante étant que les développeurs testent leur propre code
- Des développeurs ou des testeurs indépendants au sein des équipes de développement ou de l'équipe de projet ; il peut s'agir de développeurs qui testent les produits de leurs collègues
- Une équipe de test indépendante ou un groupe de testeurs au sein de l'organisation, rapportant à la direction du projet ou à la direction générale
- Des testeurs indépendants appartenant à l'organisation métier ou à la communauté d'utilisateurs, , ou spécialisés dans des types de tests spécifiques tels que l'utilisabilité, la sécurité, la performance, la conformité réglementaire ou la portabilité
- Des testeurs indépendants externes à l'organisation, travaillant soit sur site (insourcing) soit hors site (outsourcing)

Pour la plupart des projets, il est généralement préférable d'avoir plusieurs niveaux de test, certains de ces niveaux étant gérés par des testeurs indépendants. Les développeurs devraient participer aux tests, en particulier aux niveaux inférieurs, afin d'exercer un contrôle sur la qualité de leur propre travail.

La façon dont l'indépendance des tests est mise en œuvre varie en fonction du modèle de cycle de vie de développement logiciel. Par exemple, dans le développement Agile, les testeurs peuvent faire partie d'une équipe de développement. Dans certaines organisations utilisant des méthodes Agiles, il est envisageable que ces testeurs fassent partie d'une plus grande équipe de test indépendante. De plus, dans ces organisations, les Product Owners peuvent effectuer des tests d'acceptation pour valider les User Stories à la fin de chaque itération.

Les avantages potentiels de l'indépendance des tests incluent :

- Les testeurs indépendants sont susceptibles de détecter des types de défaillances différents par rapport aux développeurs en raison de leurs connaissances propres, de leurs approches techniques et de biais différents
- Un testeur indépendant peut vérifier, contester ou réfuter les hypothèses formulées par les parties prenantes au cours de la spécification et de l'implémentation du système

Les inconvénients potentiels de l'indépendance des tests sont les suivants :

- Les testeurs peuvent être isolés de l'équipe de développement, ce qui entraîne un manque de collaboration, des retards dans la communication des retours d'information à l'équipe de développement ou une relation conflictuelle avec l'équipe de développement
- Les développeurs peuvent perdre le sens des responsabilités vis-à-vis de la qualité

- Les testeurs indépendants peuvent être considérés comme un goulot d'étranglement ou être rendus responsables des retards dans la sortie de la version
- Les testeurs indépendants peuvent ne pas disposer de certaines informations importantes (par exemple, sur l'objet de test)

De nombreuses organisations réussissent à tirer parti des avantages de l'indépendance des tests tout en évitant les inconvénients.

5.1.2 Tâches d'un Test Manager et d'un testeur

Dans ce syllabus, deux rôles de test sont couverts, les Test Managers et les testeurs. Les activités et les tâches accomplies par ces deux rôles dépendent du contexte du projet et du produit, des compétences des personnes dans ces rôles et de l'organisation.

Le Test Manager porte la responsabilité globale du processus de test et de la bonne conduite des activités de test. La gestion des tests peut être réalisée par un Test Manager professionnel ou par un chef de projet, un responsable du développement ou un responsable de l'assurance qualité. Dans les grands projets ou organisations, plusieurs équipes de test peuvent être sous la responsabilité d'un Test Manager, d'un coach de test ou d'un coordinateur de test, chaque équipe étant dirigée par un Test Leader (appelé aussi Test Lead).

Les tâches habituelles du Test Manager peuvent inclure :

- Développer ou revoir une politique de test et une stratégie de test pour l'organisation
- Planifier les activités de test en tenant compte du contexte, y compris les objectifs et les risques du test. Cela peut inclure la sélection des approches de test, l'estimation du temps, de l'effort et du coût des tests, l'acquisition de ressources, la définition des niveaux de test et des cycles de test et la planification de la gestion des défauts
- Rédiger et mettre à jour le(s) plan(s) de test
- Coordonner le(s) plan(s) de test avec les chefs de projet, les Product Owners et d'autres parties prenantes
- Partager différents aspects des tests avec d'autres activités du projet, comme la planification de l'intégration
- Lancer l'analyse, la conception, l'implémentation et l'exécution des tests, suivre l'avancement et les résultats des tests, et vérifier le statut des critères de sortie (ou de la définition du terminé)
- Préparer et fournir des rapports d'avancement des tests et des rapports de synthèse des tests basés sur les informations recueillies
- Adapter la planification en fonction des résultats et de l'avancement des tests (parfois documentés dans les rapports d'avancement des tests et/ou dans les rapports de synthèse des tests pour d'autres tests déjà réalisés sur le projet) et prendre toutes les mesures nécessaires au contrôle des tests
- Aider la mise en place du système de gestion des défauts et à la gestion adéquate de la configuration des testware
- Introduire les métriques appropriées pour mesurer l'avancement des tests et évaluer la qualité des tests et du produit

- Accompagner la sélection et la mise en œuvre des outils pour soutenir le processus de test, y compris recommander le budget pour la sélection des outils (et éventuellement l'achat et/ou le support), allouer du temps et des ressources pour des projets pilotes et apporter un accompagnement continu à l'utilisation du ou des outils
- Décider de l'implémentation du ou des environnements de test
- Promouvoir et soutenir les testeurs, l'équipe de test et le métier du test au sein de l'organisation
- Développer les compétences et les carrières des testeurs (par exemple, par le biais de plans de formation, d'évaluations de performance, de coaching, etc.).

La façon dont le rôle de Test Manager est exécuté varie en fonction du cycle de vie du développement du logiciel. Par exemple, dans le développement Agile, certaines des tâches mentionnées ci-dessus sont prises en charge par l'équipe Agile, en particulier les tâches liées aux tests quotidiens réalisés au sein de l'équipe, souvent par un testeur travaillant au sein de l'équipe. Certaines des tâches qui s'étendent sur plusieurs équipes ou sur l'ensemble de l'organisation, ou qui ont un rapport avec la gestion du personnel, peuvent être effectués par des Test Managers à l'extérieur de l'équipe de développement, qui sont parfois appelés coach de tests. Voir Black 2009 pour plus d'informations sur la gestion du processus de test.

Les tâches habituelles des testeurs comprennent notamment :

- Revoir et contribuer aux plans de test
- Analyser, revoir et évaluer les exigences, les User Stories et les critères d'acceptation, les spécifications et les modèles (c.-à-d. les bases de test) vis à vis de leur testabilité
- Identifier et documenter les conditions de test, et saisir la traçabilité entre les cas de test, les conditions de test et les bases de test
- Concevoir, configurer et vérifier le ou les environnement(s) de test, souvent en se coordonnant avec l'administration système et réseau
- Concevoir et implémenter les cas de test et les procédures de test
- Préparer et acquérir des données de test
- Créer le planning détaillé d'exécution des tests
- Exécuter des tests, évaluer les résultats et documenter les écarts par rapport aux résultats attendus
- Utiliser les outils appropriés pour faciliter le processus de test
- Automatiser les tests selon les besoins (peut être porté par un développeur ou un expert en automatisation des tests)
- Évaluer les caractéristiques non-fonctionnelles telles que la performance, la fiabilité, l'utilisabilité, la sécurité, la compatibilité et la portabilité
- Revoir des tests développés par d'autres

Les personnes qui travaillent à l'analyse des tests, à la conception des tests, à des types de tests spécifiques ou à l'automatisation des tests peuvent être des spécialistes dans ces rôles. En fonction des risques produit et projet, et du modèle de cycle de vie du développement logiciel choisi, différentes personnes peuvent assumer le rôle de testeur à différents niveaux de test. Par exemple, au niveau du test des composants et du test d'intégration des composants, le rôle de testeur est souvent porté par des développeurs. Au niveau du test d'acceptation, le rôle de testeur est souvent porté par des analystes

métier, des experts du domaine et des utilisateurs. Aux niveaux du test du système et du test d'intégration de systèmes, le rôle de testeur est souvent porté par une équipe de test indépendante. Au niveau des tests d'acceptation opérationnelle, le rôle de testeur est souvent porté par l'équipe d'exploitation et/ou d'administration.

5.2 Planification et estimation des tests

5.2.1 Objet et contenu d'un plan de test

Un plan de test décrit les activités de test pour des projets de développement et de maintenance. La planification est influencée par la politique et la stratégie de test de l'organisation, les cycles de vie du développement et les méthodes utilisées (voir section 2.1), l'étendue des tests, les objectifs, les risques, les contraintes, la criticité, la testabilité et la disponibilité des ressources.

Au fur et à mesure que le projet et la planification des tests progressent, davantage d'informations deviennent disponibles et plus de détails peuvent être inclus dans le plan de test. La planification des tests est une activité continue et s'effectue tout au long du cycle de vie du produit. (Notez que le cycle de vie du produit peut s'étendre au-delà de la portée d'un projet pour inclure la phase de maintenance). Des retours d'information sur les activités de test devraient être utilisés pour identifier les risques qui évoluent afin que la planification puisse être ajustée. La planification peut être documentée dans un plan de test maître et dans des plans de test séparés pour des niveaux de test, tels que les tests système et les tests d'acceptation, ou pour des types de test séparés, tels que les tests d'utilisabilité et les tests de performance. Les activités de planification des tests peuvent comprendre les éléments suivants, dont certains peuvent être documentés dans un plan de test :

- Déterminer le périmètre, les objectifs et les risques des tests
- Définir l'approche générale des tests
- Intégrer et coordonner les activités de test dans les activités du cycle de vie du logiciel
- Prendre des décisions sur ce qu'il faut tester, les personnes et les autres ressources nécessaires pour effectuer les diverses activités de test et la façon dont les activités de test seront effectuées
- Planifier les activités d'analyse, de conception, d'implémentation, d'exécution et d'évaluation des tests, soit à des dates particulières (p. ex., en développement séquentiel) ou dans le contexte de chaque itération (p. ex., en développement itératif)
- Sélectionner les métriques pour le pilotage et le contrôle des tests
- Budgéter les activités de test
- Déterminer le niveau de détail et la structure de la documentation des tests (p. ex. en fournissant des modèles ou des exemples de documents)

Le contenu des plans de test varie et peut s'étendre au-delà des sujets identifiés ci-dessus. Des exemples de plans de test peuvent être trouvés dans la norme ISO (ISO/IEC/IEEE 29119-3).

5.2.2 Stratégie de test et approche de test

Une stratégie de test fournit une description générale du processus de test, généralement au niveau du produit ou de l'organisation. Les types les plus courants de stratégies de test sont les suivants :

- **Analytique** : Ce type de stratégie de test est basé sur l'analyse d'un facteur (p. ex., exigences ou risques). Le test basé sur les risques est un exemple d'approche analytique, où les tests sont conçus et priorisés en fonction du niveau de risque.
- **Basée sur des modèles** : Dans ce type de stratégie de test, les tests sont conçus à partir d'un modèle d'un aspect requis du produit, comme une fonction, un processus métier, une structure interne ou une caractéristique non-fonctionnelle (par exemple, la fiabilité). Les modèles de processus métier, les modèles d'état et les modèles de fiabilité en sont des exemples.
- **Méthodique** : Ce type de stratégie de test repose sur l'utilisation systématique d'un ensemble prédéfini de tests ou de conditions de test, tels qu'une taxonomie des types de défaillances les plus courantes ou probables, une liste des caractéristiques de qualité importantes, ou des normes d'entreprises relatives à l'aspect à la convivialité des applications mobiles ou des pages Web.
- **Conforme à un processus (ou conforme à une norme)** : Ce type de stratégie de test implique l'analyse, la conception et l'implémentation de tests basés sur des règles et normes externes, telles que celles spécifiées par des normes spécifiques à l'industrie, par la documentation des processus, par l'identification et l'utilisation rigoureuse de la base de test, ou par tout processus ou norme imposée à l'organisation ou par l'organisation.
- **Dirigée (ou consultative)** : Ce type de stratégie de test est principalement dicté par les recommandations, les conseils ou les instructions des parties prenantes, des experts du domaine métier ou des experts techniques, qui peuvent se trouver en dehors de l'équipe de test ou de l'organisation elle-même.
- **Anti-régressions** : Ce type de stratégie de test est motivé par le désir d'éviter la régression au niveau des fonctionnalités existantes. Cette stratégie de test comprend la réutilisation des testware existants (en particulier les cas de test et les données de test), une forte automatisation des tests de régression et des suites de test standard.
- **Réactive** : Dans ce type de stratégie de test, le test est adapté aux composants ou systèmes testés en réaction aux événements se produisant pendant l'exécution des tests, plutôt que pré-planifié (comme le prévoient les stratégies précédentes). Les tests sont conçus et implémentés, et peuvent être exécutés immédiatement en réponse aux informations obtenues à partir des résultats de tests antérieurs. Les tests exploratoires sont une technique couramment employée dans les stratégies réactives.

Une stratégie de test pertinente est souvent créée en combinant plusieurs de ces types de stratégies de test. Par exemple, le test basé sur les risques (une stratégie analytique) peut être combiné avec des tests exploratoires (une stratégie réactive) ; ils se complètent mutuellement et peuvent permettre d'obtenir des tests plus efficaces lorsqu'ils sont utilisés ensemble.

Alors que la stratégie de test fournit une description générale du processus de test, l'approche de test ajuste la stratégie de test pour un projet ou une version particulière. L'approche de test est le point de départ pour sélectionner les techniques de test, les niveaux de test et les types de test, et pour définir les critères d'entrée et de sortie (ou définition du prêt et définition du terminé, respectivement). L'ajustement de la stratégie est basé sur les décisions prises en fonction de la complexité et des objectifs du projet, du type de produit développé et de l'analyse des risques produit. L'approche choisie dépend du contexte et peut tenir compte de facteurs tels que les risques, la sécurité, les ressources et les compétences disponibles, la technologie, la nature du système (p. ex., construit sur mesure versus sur étagère), les objectifs des tests et les réglementations.

5.2.3 Critères d'entrée et de sortie (Définition du prêt et définition du terminé)

Afin d'exercer un contrôle efficace sur la qualité du logiciel et des tests, il est conseillé d'avoir des critères qui définissent quand une activité de test donnée doit commencer et quand l'activité est terminée. Les critères d'entrée (plus typiquement appelés définition du prêt dans le développement Agile) définissent les préconditions pour entreprendre une activité de test donnée. Si les critères d'entrée ne sont pas satisfaits, il est probable que l'activité s'avérera plus difficile, plus longue, plus coûteuse et plus risquée. Les critères de sortie (plus typiquement appelés définition du terminé dans le développement Agile) définissent les conditions à remplir pour pouvoir déclarer un niveau de test ou un ensemble de tests terminés. Les critères d'entrée et de sortie devraient être définis pour chaque niveau et type de test, et différeront en fonction des objectifs de test.

Les critères d'entrée habituels incluent :

- Disponibilité d'exigences testables, de User Stories et/ou de modèles (par exemple, lorsque l'on suit une stratégie de test basée sur les modèles)
- Disponibilité d'éléments de test qui ont satisfait aux critères de sortie pour tous les niveaux de test précédents
- Disponibilité de l'environnement de test
- Disponibilité des outils de test nécessaires
- Disponibilité des données de test et autres ressources nécessaires

Les critères de sortie habituels incluent :

- Les tests planifiés ont été exécutés
- Un niveau défini de couverture (par exemple, des exigences, des User Stories, des critères d'acceptation, des risques, du code) a été atteint
- Le nombre de défauts non résolus est limité à ce qui a été défini
- Le nombre estimé de défauts restants est suffisamment faible
- Les niveaux évalués de fiabilité, de performance, d'utilisabilité, de sécurité et autres caractéristiques qualité pertinentes sont suffisants

Même si les critères de sortie ne sont pas satisfaits, il est également courant que les activités de test soient réduites en raison du budget dépensé, du temps prévu et/ou de la pression pour mettre le produit sur le marché. Il peut être acceptable de mettre fin aux tests dans de telles circonstances, si les parties prenantes du projet et les responsables métier ont revu et accepté le risque de mettre en production sans autres tests.

5.2.4 Calendrier d'exécution des tests

Une fois que les différents cas de test et procédures de test sont produits (avec des procédures de test potentiellement automatisées) et assemblés en suites de test, les suites de test peuvent être organisées selon un calendrier d'exécution des tests qui définit l'ordre dans lequel elles doivent être exécutées. Le calendrier d'exécution des tests devrait tenir compte de facteurs tels que l'ordre de priorité, les dépendances, les tests de confirmation, les tests de régression et la séquence la plus efficace pour exécuter les tests.

Idéalement, les cas de test devraient être exécutés en fonction de leur niveau de priorité, généralement en exécutant en premier les cas de test ayant la priorité la plus élevée. Cependant, cette pratique peut ne

pas fonctionner si les cas de test ont des dépendances ou si les caractéristiques testées ont des dépendances. Si un cas de test avec une priorité plus élevée dépend d'un cas de test avec une priorité plus faible, le cas de test de priorité plus faible doit être exécuté en premier. De même, s'il existe des dépendances entre les cas de test, il faut les ordonner de façon appropriée, indépendamment de leurs priorités relatives. Les tests de confirmation et de régression doivent également être priorisés, en fonction de l'importance d'une remontée d'information rapide sur les changements réalisés, mais là encore, des dépendances peuvent s'appliquer.

Dans certains cas, divers enchaînements de tests sont possibles, avec différents niveaux d'efficacité associés à ces enchaînements. Dans ce cas, il faut trouver un compromis entre l'efficacité de l'exécution des tests et le respect des priorités.

5.2.5 Facteurs influençant l'effort de test

L'estimation de l'effort de test consiste à prédire la quantité de travail lié aux tests qui sera nécessaire pour atteindre les objectifs de test d'un projet, d'une version ou d'une itération particulière. Les facteurs qui influencent l'effort de test peuvent inclure les caractéristiques du produit, les caractéristiques du processus de développement, les caractéristiques liées aux personnes et les résultats des tests, comme détaillé ci-dessous.

Caractéristiques du produit

- Les risques associés au produit
- La qualité des bases de test
- La taille du produit
- La complexité du domaine métier du produit
- Les exigences relatives aux caractéristiques de qualité (p. ex., sécurité, fiabilité)
- Le niveau de détail requis pour la documentation des tests
- Les exigences en matière de conformité juridique et réglementaire

Caractéristiques du processus de développement

- La stabilité et la maturité de l'organisation
- Le modèle de développement utilisé
- L'approche de test
- Les outils utilisés
- Le processus de test
- La pression des délais

Caractéristiques liées aux personnes

- Les compétences et l'expérience des personnes impliquées, en particulier avec des projets et des produits similaires (par exemple, connaissance du domaine)
- La cohésion et le leadership de l'équipe

Résultats des tests

- Le nombre et la sévérité des défauts trouvés

- La quantité nécessaire de travail à refaire

5.2.6 Techniques d'estimation des tests

Il existe un certain nombre de techniques d'estimation utilisées pour déterminer l'effort requis pour des tests adéquats. Deux des techniques les plus couramment utilisées sont les suivantes :

- La technique basée sur les métriques : estimer l'effort de test sur la base de métriques d'anciens projets similaires ou sur la base de valeurs représentatives
- La technique basée sur l'expertise : estimation de l'effort de test sur la base de l'expérience des responsables des tâches de test ou par des experts

Par exemple, dans le développement Agile, les burndown charts sont des exemples d'une approche basée sur les métriques car l'effort est recueilli et enregistré, et est ensuite utilisé pour déterminer la quantité de travail que l'équipe peut faire dans la prochaine itération ; alors que le planning poker est un exemple d' approche basée sur l'expertise, car les membres de l'équipe estiment l'effort pour livrer une fonctionnalité en se basant sur leur expérience (cf. syllabus Testeur Certifié - Extension Niveau Fondation Testeur Agile CFTL / ISTQB).

Dans les projets séquentiels, les modèles de correction des défauts sont des exemples de l'approche basée sur les métriques, où des volumes de défauts et le temps nécessaire pour les éliminer sont recueillis et enregistrés, ce qui fournit ensuite une base pour estimer les projets futurs de nature similaire ; tandis que la technique d'estimation Wideband Delphi est un exemple d'approche basée sur l'expertise dans laquelle des groupes d'experts fournissent des estimations basées sur leur expérience (cf. au syllabus CFTL / ISTQB Testeur Certifié de Niveau Avancé Test Manager).

5.3 Pilotage et contrôle des tests

Le but du pilotage des tests est de recueillir des informations et de fournir un retour et de la visibilité sur les activités de test. Les informations à suivre peuvent être collectées manuellement ou automatiquement et devraient être utilisées pour évaluer l'avancement du test et pour mesurer si les critères de sortie du test, ou les tâches de test associées à la définition du terminé d'un projet Agile, sont satisfaits, par exemple le respect des objectifs de couverture des risques produit, des exigences ou des critères d'acceptation.

Le contrôle des tests décrit toutes les mesures correctives ou d'orientation prises à la suite de l'information et des métriques recueillies et (éventuellement) rapportées. Les actions peuvent couvrir toutes les activités de test et peuvent affecter toute autre activité du cycle de vie du logiciel.

Voici des exemples d'actions de contrôle des tests :

- Re-prioriser les tests lorsqu'un risque identifié se produit (p. ex. logiciel livré en retard)
- Modifier le calendrier des tests en raison de la disponibilité ou de l'indisponibilité d'un environnement de test ou d'autres ressources
- Réévaluer si un élément de test répond à un critère d'entrée ou de sortie à cause d'une modification

5.3.1 Métriques utilisées pour les tests

Des métriques peuvent être recueillies pendant et à la fin des activités de test afin d'évaluer :

- Avancement par rapport au calendrier et au budget prévus

- Qualité actuelle de l'objet de test
- Pertinence de l`approche de test
- Efficacité des activités de test par rapport aux objectifs

Les métriques de test les plus courantes sont les suivantes :

- Pourcentage du temps de travail prévu réalisé pour la préparation des cas de test (ou le pourcentage des cas de test prévus implémentés)
- Pourcentage du travail prévu réalisé pour la préparation de l'environnement de test
- Exécution des cas de test (p. ex. nombre de cas de test exécutés/non exécutés, cas de test réussis/échoués, et/ou conditions de test réussies/échouées)
- Informations sur les défauts (p. ex. densité des défauts, défauts trouvés et corrigés, taux de défaillance et résultats des tests de confirmation)
- Couverture des exigences, des User Stories, des critères d'acceptation, des risques ou du code par les tests
- Degré d'achèvement des tâches, affectation et utilisation des ressources, et temps passé
- Coût des tests, y compris en comparaison avec le bénéfice apporté par la découverte des défauts supplémentaires ou le coût par rapport au bénéfice de l'exécution de tests supplémentaires

5.3.2 Buts, contenu et destinataires des rapports de test

Le but du rapport de test est de synthétiser et de communiquer les informations sur l'activité de test, à la fois pendant et à la fin d'une activité de test (par exemple, un niveau de test). Le rapport de test préparé pendant une activité de test peut être appelé rapport d'avancement de test, tandis qu'un rapport de test préparé à la fin d'une activité de test peut être appelé rapport de synthèse de test.

Au cours du pilotage et du contrôle des tests, le Test Manager publie régulièrement des rapports d'avancement de test pour les parties prenantes. En plus du contenu commun aux rapports d'avancement de test et aux rapports de synthèse de test, les rapports d'avancement de test peuvent également inclure les éléments suivants :

- Le statut des activités de test et l'avancement par rapport au plan de test
- Les facteurs qui freinent l'avancement
- Les tests prévus pour la prochaine période de suivi
- La qualité de l'objet de test

Lorsque les critères de sortie sont atteints, le Test Manager fournit le rapport de synthèse de test. Ce rapport fournit un résumé des tests réalisés, sur la base du dernier rapport d'avancement de test et de toute autre information pertinente.

Les rapports d'avancement de test et les rapports de synthèse de test peuvent comprendre les éléments suivants :

- Récapitulatif des tests réalisés
- Informations sur le déroulement de la période de test

- Ecarts par rapport au plan, y compris les écarts par rapport au calendrier, à la durée ou à l'effort consacré aux activités de test
- Statut des tests et de la qualité du produit par rapport aux critères de sortie ou à la définition du terminé
- Facteurs ayant bloqué ou bloquant l'avancement
- Métriques sur les défauts, les cas de test, la couverture de test, la l'avancement de l'activité et la consommation de ressources. (p. ex., tel que décrit au point 5.3.1)
- Risques résiduels (voir section 5.5)
- Produits d'activités de test réutilisables

Le contenu d'un rapport de test variera en fonction du projet, des exigences organisationnelles et du cycle de vie du développement logiciel. Par exemple, un projet complexe avec de nombreuses parties prenantes ou un projet réglementé peuvent exiger des rapports plus détaillés et plus rigoureux qu'une mise à jour rapide du logiciel. Autre exemple, dans le développement Agile, les rapports d'avancement de test peuvent être incorporés dans les tableaux de tâches, les récapitulatifs de défauts et les burndown charts, qui peuvent être discutés lors d'une réunion quotidienne (cf. syllabus Testeur Certifié - Extension Niveau Fondation Testeur Agile CFTL / ISTQB).

En plus d'ajuster les rapports de test en fonction du contexte du projet, les rapports de test devraient aussi être ajustés en fonction de l'audience du rapport. Le type et la quantité d'informations à inclure pour un destinataire technique ou une équipe de test peuvent être différents de ce qui serait inclus dans un rapport de synthèse résumé. Dans le premier cas, des informations détaillées sur les types de défauts et les évolutions tendancielles peuvent être importantes. Dans le second cas, un rapport de haut niveau (p. ex. un résumé de l'état des défauts par priorité, budget, calendrier et conditions de test réussies/échouées/ non testées) peut être plus approprié.

La norme ISO (ISO/IEC/IEEE 29119-3) fait référence à deux types de rapports de test, les rapports d'avancement de test et les rapports de fin de test (appelés rapports de synthèse de test dans ce syllabus), et contient les structures et les exemples pour chaque type.

5.4 Gestion de configuration

Le but de la gestion de la configuration est d'établir et de maintenir l'intégrité du composant ou du système, du testware et de leurs relations mutuelles tout au long du cycle de vie du projet et du produit.

Pour supporter correctement les tests, la gestion de la configuration peut impliquer de s'assurer de ce qui suit :

- Tous les éléments de test sont identifiés de façon unique, versionnés, suivis pour les changements et reliés les uns aux autres
- Tous les éléments du testware sont identifiés de manière unique, versionnés, suivis pour les changements, liés les uns aux autres et liés aux versions des éléments de test afin que la traçabilité puisse être maintenue tout au long du processus de test
- Tous les documents et logiciels identifiés sont référencés sans ambiguïté dans la documentation de test

Au cours de la planification des tests, les procédures de gestion de la configuration et l'infrastructure (outils) devraient être identifiées et implémentées.

5.5 Risques et tests

5.5.1 Définition du risque

Le risque implique la possibilité d'un événement futur qui a des conséquences négatives. Le niveau de risque est déterminé par la probabilité de l'événement et son impact (le préjudice).

5.5.2 Risques produit et risques projet

Le risque produit implique la possibilité qu'un produit d'activités (par exemple, une spécification, un composant, un système ou un test) puisse ne pas satisfaire les besoins de ses utilisateurs et/ou parties prenantes. Lorsque les risques produit sont associés à des caractéristiques de qualité spécifiques d'un produit (p. ex. adéquation fonctionnelle, fiabilité, performance, utilisabilité, sécurité, compatibilité, maintenabilité et portabilité), les risques produit sont également appelés risques qualité. Voici des exemples de risques produit :

- Le logiciel peut ne pas remplir les fonctions attendues selon la spécification
- Le logiciel peut ne pas remplir les fonctions attendues selon les besoins de l'utilisateur, du client et/ou des parties prenantes
- Une architecture système peut ne pas répondre de manière adéquate à certaines exigences non-fonctionnelles
- Un traitement particulier peut être effectué de manière incorrecte dans certaines circonstances
- Une structure de contrôle de boucle peut être mal codée
- Les délais de réponse peuvent être insatisfaisants pour un système de traitement des transactions à haut niveau de performance
- L'expérience utilisateur (UX - User eXperience) pourrait ne pas satisfaire les attentes pour le produit

Le risque projet implique des situations qui, si elles se produisent, peuvent avoir un effet négatif sur la capacité d'un projet à atteindre ses objectifs. Voici des exemples de risques projet :

- Problèmes liés au projet :
  - Des retards peuvent se produire dans la livraison, l'achèvement des tâches ou la satisfaction des critères de sortie ou de la définition du terminé
  - Des estimations erronées, la réaffectation de ressources à des projets plus prioritaires ou une réduction générale des coûts dans l'ensemble de l'organisation peuvent entraîner un financement inadéquat
  - Des changements tardifs peuvent entraîner un travail supplémentaire important
- Problèmes organisationnels :
  - Les compétences, la formation et le nombre de personnes peuvent ne pas être suffisants
  - Les problèmes personnels peuvent causer des conflits et des problèmes
  - Les utilisateurs, les personnes du métier ou les experts du domaine peuvent ne pas être disponibles en raison de priorités contradictoires
- Problèmes politiques :

- Les testeurs peuvent ne pas communiquer leurs besoins et/ou les résultats des tests de manière adéquate
- Les développeurs et/ou les testeurs peuvent ne pas assurer le suivi des informations trouvées dans les tests et les revues (par exemple, ne pas améliorer les pratiques de développement et de test)
- Il peut y avoir une attitude ou des attentes erronées vis-à-vis des tests (p. ex. ne pas apprécier la valeur de la détection des défauts pendant les tests)

- Questions techniques :
  - Les exigences peuvent ne pas être suffisamment bien définies
  - Les exigences peuvent ne pas être satisfaites, compte tenu des contraintes existantes
  - L'environnement de test peut ne pas être prêt à temps
  - La conversion des données, la planification de la migration et leur outillage peuvent être en retard
  - Les faiblesses dans le processus de développement peuvent avoir un impact sur la cohérence ou la qualité des produits d'activités du projet tels que la conception, le code, la configuration, les données de test et les cas de test
  - Une mauvaise gestion des défauts et des problèmes similaires peut entraîner une accumulation de défauts et d'autres dettes techniques
- Problèmes liés aux fournisseurs :
  - Un tiers peut ne pas livrer un produit ou un service nécessaire, ou faire faillite
  - Des problèmes contractuels peuvent causer des difficultés au projet

Les risques Projet peuvent affecter à la fois les activités de développement et les activités de test. Dans certains cas, les chefs de projet sont responsables de la gestion de tous les risques du projet, mais il n'est pas inhabituel que les Test Managers soient responsables des risques projet en lien avec les tests.

5.5.3 Test basé sur les risques et qualité du produit

Le risque est utilisé pour orienter l'effort requis pendant les tests. Il est utilisé pour décider où et quand commencer les tests et pour identifier les domaines qui nécessitent plus de vigilance. Les tests sont utilisés pour réduire la probabilité qu'un événement indésirable se produise ou pour réduire l'impact d'un événement indésirable. Les tests sont utilisés comme activité d'atténuation des risques, afin de fournir une remontée d'information sur les risques identifiés, ainsi qu'une restitution sur les risques résiduels (non résolus).

Une approche de test basé sur les risques offre des possibilités proactives de réduire les niveaux de risques produit. Elle comprend l'analyse des risques produit, qui comprend l'identification des risques produit et l'évaluation de la probabilité et de l'impact de chaque risque. L'information sur le risque Produit qui en résulte est utilisée pour guider la planification des tests, la spécification, la préparation et l'exécution des cas de test, ainsi que le pilotage et le contrôle des tests. L'analyse précoce des risques produit contribue au succès d'un projet.

Dans une approche basée sur les risques, les résultats de l'analyse du risque produit sont utilisés pour :

- Déterminer les techniques de test à utiliser

- Déterminer les niveaux et les types spécifiques de tests à effectuer (p. ex. tests de sécurité, tests d'accessibilité)
- Déterminer le volume des tests à réaliser
- Prioriser les tests afin de trouver les défauts critiques le plus tôt possible
- Déterminer si des activités en complément des tests pourraient être utilisées pour réduire les risques (p. ex. formation de concepteurs inexpérimentés)

Le test basé sur les risques s'appuie sur les connaissances et les observations des parties prenantes du projet pour effectuer l'analyse des risques produit. Pour s'assurer que la probabilité de défaillance d'un produit est réduite au minimum, les activités de gestion des risques fournissent une approche méthodique pour :

- Analyser (et réévaluer régulièrement) ce qui peut mal se passer (risques)
- Déterminer quels risques sont importants à gérer
- Mettre en œuvre des mesures pour atténuer ces risques
- Établir des plans de contingence pour faire face aux risques si ceux-ci devenaient des événements effectifs

De plus, les tests peuvent identifier de nouveaux risques, aider à déterminer quels risques devraient être atténués et réduire l'incertitude au sujet des risques.

5.6 Gestion des défauts

Comme l'un des objectifs des tests est de trouver des défauts, les défauts trouvés pendant les tests devraient être enregistrés. La façon dont les défauts sont enregistrés peut varier en fonction du contexte du composant ou du système testé, du niveau de test et du modèle de cycle de vie de développement logiciel. Tout défaut identifié doit faire l'objet d'une analyse et faire l'objet d'un suivi depuis sa découverte et sa classification jusqu'à sa résolution (p. ex. correction des défauts et tests de confirmation de la solution réussi, report à une version ultérieure, acceptation en tant que limitation permanente du produit, etc.). Afin de gérer tous les défauts jusqu'à la résolution, une organisation devrait définir un processus de gestion des défauts qui comprend un workflow et des règles de classification. Ce processus doit être approuvé avec tous ceux qui participent à la gestion des défauts, y compris les concepteurs, les développeurs, les testeurs et les Product Owners. Dans certaines organisations, l'enregistrement et le suivi des défauts peuvent être très informels.

Au cours du processus de gestion des défauts, certains rapports peuvent s'avérer contenir des faux positifs, et non des défaillances réelles dues à des défauts. Par exemple, un test peut échouer lorsqu'une connexion réseau est interrompue ou a un temps de réponse trop long. Ce comportement ne résulte pas d'un défaut dans l'objet de test, mais est une anomalie qui doit être examinée. Les testeurs devraient faire en sorte de réduire au minimum le nombre de faux positifs signalés comme étant des défauts.

Les défauts peuvent être rapportés pendant le codage, l'analyse statique, les revues, les tests dynamiques ou l'utilisation d'un produit logiciel. Les défauts peuvent être rapportés pour des problèmes dans le code ou pour des systèmes en utilisation, ou dans tout type de documentation, y compris les exigences, les User Stories et les critères d'acceptation, les documents de développement, les documents de test, les manuels d'utilisation ou les guides d'installation. Afin d'avoir un processus de gestion des défauts efficace et efficient, les organisations peuvent définir des normes concernant les attributs, la classification et le workflow des défauts.

Les rapports de défauts ont généralement les objectifs suivants :

- Fournir aux développeurs et aux autres intervenants de l'information sur tout événement indésirable survenu, afin de leur permettre de déterminer les effets spécifiques, d'isoler le problème avec un test de reproduction à minima et de corriger le ou les défauts potentiels, au besoin, ou de résoudre le problème d'une autre manière
- Fournir aux Test Managers un moyen de suivre la qualité du produit d'activités testé et l'impact sur les tests (par exemple, si un grand nombre de défauts sont signalés, les testeurs auront passé beaucoup de temps à les signaler au lieu d'effectuer des tests, et il y aura plus de tests de confirmation nécessaires)
- Fournir des idées pour le développement et l'amélioration du processus de test

Un rapport de défaut rempli pendant les tests dynamiques comprend généralement les éléments suivants :

- Un identifiant
- Un titre et un bref résumé du défaut signalé
- La date du rapport de défaut, l'organisation émettrice et l'auteur
- L'identification de l'élément de test (élément de configuration testé) et de l'environnement
- La ou les phases du cycle de développement au cours desquelles le défaut a été observé
- Une description du défaut pour permettre la reproduction et la résolution, y compris des logs, des captures de base de données, des captures d'écran ou des enregistrements (si trouvés pendant l'exécution des tests)
- Les résultats attendus et obtenus
- La portée ou le degré d'impact (sévérité) du défaut pour les parties prenantes concernées
- Urgence/priorité de la correction
- L'état du rapport de défaut (p. ex. ouvert, différé, en double, en attente de correction, en attente de test de confirmation, ré-ouvert, fermé)
- Les conclusions, recommandations et approbations
- Les enjeux généraux, tels que les autres domaines qui peuvent être affectés par un changement résultant du défaut
- L'historique des modifications, notamment la séquence des mesures prises par les membres de l'équipe de projet quant au défaut pour isoler, réparer et confirmer qu'il est corrigé
- Les références, y compris le cas de test qui a révélé le problème

Certains de ces détails peuvent être automatiquement inclus et/ou gérés lors de l'utilisation d'outils de gestion des défauts, par exemple, l'attribution automatique d'un identifiant, l'attribution et la mise à jour de l'état du rapport de défaut pendant le workflow, etc. Les défauts détectés au cours des tests statiques, en particulier les revues, seront normalement documentés d'une manière différente, par exemple, dans les notes de réunion de revue.

Un exemple du contenu d'un rapport de défaut peut être trouvé dans la norme ISO (ISO/IEC/IEEE 29119-3) (qui se réfère aux rapports de défaut en termes de rapports d'incidents).

6 Outils de support aux tests — 40 minutes

Termes

test piloté par les données, test piloté par les mots-clés, outil de test de performance, automatisation des tests, outil d'exécution des tests, outil de gestion des tests

Objectifs d'apprentissage pour les outils de support aux tests

6.1 Introduction aux outils de test

FL-6.1.1 (K2) Classer les outils de test selon leur objet et les activités de test prises en charge

FL-6.1.2 (K1) Identifier les bénéfices et les risques de l'automatisation des tests

FL-6.1.3 (K1) Se souvenir des considérations particulières pour les outils d'exécution des tests et les outils de gestion des tests

6.2 Utilisation efficace des outils

FL-6.2.1 (K1) Identifier les principes généraux pour la sélection d'un outil

FL-6.2.2 (K1) Rappeler les objectifs de l'utilisation de projets pilotes pour introduire des outils

FL-6.2.3 (K1) Identifier les facteurs de succès pour l'évaluation, l'implémentation, le déploiement et le support continu, des outils de tests dans une organisation

6.1 Introduction aux outils de test

Les outils de test peuvent être utilisés pour soutenir une ou plusieurs activités de test. Ces outils comprennent :

- Les outils directement utilisés dans les tests, tels que les outils d'exécution des tests et les outils de préparation des données de test
- Les outils qui aident à gérer les exigences, les cas de test, les procédures de test, les scripts de test automatisés, les résultats de test, les données de test et les défauts, ainsi que pour le reporting et le pilotage de l'exécution des tests
- Les outils utilisés pour la recherche et l'évaluation
- Tout outil d'aide aux tests (une feuille de calcul est aussi un outil de test dans ce sens)

6.1.1 Classification des outils de test

Les outils de test peuvent avoir une ou plusieurs des finalités suivantes en fonction du contexte :

- Améliorer l'efficacité des activités de test en automatisant des tâches répétitives ou des tâches qui nécessitent des ressources importantes lorsqu'elles sont effectuées manuellement (p. ex. exécution de test, tests de régression)
- Améliorer l'efficacité des activités de test en assistant les activités de test manuelles tout au long du processus de test (voir section 1.4)
- Améliorer la qualité des activités de test en permettant des tests plus cohérents et un niveau plus élevé de reproductibilité des défauts
- Automatiser les activités qui ne peuvent être exécutées manuellement (p. ex. tests de performance à grande échelle)
- Augmenter la fiabilité des tests (par exemple, en automatisant des comparaisons de données importantes ou en simulant un comportement)

Les outils peuvent être classés en fonction de plusieurs critères tels que l'objet, le prix, le modèle de licence (p. ex., commercial ou open source) et la technologie utilisée. Les outils sont classés dans ce syllabus en fonction des activités de test qu'ils supportent.

Certains outils sont dédiés totalement ou principalement à une seule activité ; d'autres peuvent soutenir plus d'une activité, mais sont classés sous l'activité à laquelle ils sont le plus étroitement associés. Les outils d'un même fournisseur, en particulier ceux qui ont été conçus pour fonctionner ensemble, peuvent être fournis sous la forme d'une suite intégrée.

Certains types d'outils de test peuvent être intrusifs, ce qui signifie qu'ils peuvent affecter le résultat effectif du test. Par exemple, les temps de réponse effectifs d'une application peuvent être différents en raison des instructions supplémentaires qui sont exécutées par un outil de test de performance, ou le montant de la couverture de code obtenu peut être faussé en raison de l'utilisation d'un outil de couverture. La conséquence de l'utilisation d'outils intrusifs s'appelle l'effet de sonde.

Certains outils offrent un service qui est plus approprié pour les développeurs (par exemple, les outils utilisés lors des tests de composants et d'intégration). Ces outils sont marqués d'un "(D)" dans les sections suivantes.

Outils pour la gestion des tests et du testware

Les outils de gestion peuvent s'appliquer à toutes les activités de test sur l'ensemble du cycle de vie de développement logiciel. Voici des exemples d'outils qui aident à la gestion des tests et du testware :

- Outils de gestion des tests et outils de gestion du cycle de vie des applications
- Outils de gestion des exigences (p. ex. traçabilité des objets de test)
- Outils de gestion des défauts
- Outils de gestion de la configuration
- Outils d'intégration continue (D)

Outils pour les tests statiques

Les outils de test statique sont associés aux activités et aux bénéfices décrits au chapitre 3. Voici des exemples d'outils de ce type :

- Outils de support aux revues
- Outils d'analyse statique (D)

Outils pour la conception et l'implémentation des tests

Les outils de conception de test aident à la création de produits d'activités maintenables lors de la conception et l'implémentation des tests, y compris les cas de test, les procédures de test et les données de test. Voici des exemples d'outils de ce type :

- Outils de conception de tests
- Outils de Model-Based Testing
- Outils de préparation des données de test
- Outils pour le développement dirigé par les tests d'acceptations (ATDD : Acceptance Test Driven Development) et le développement dirigé par le comportement (BDD : Behavior Driven Development)
- Outils pour le développement dirigé par les tests (TDD : Test Driven Development) (D)

Dans certains cas, les outils qui prennent en charge la conception et l'implémentation des tests peuvent également prendre en charge l'exécution et les logs de tests, ou fournir leurs sorties directement à d'autres outils qui prennent en charge l'exécution et les logs de tests.

Outils pour l'exécution et les logs de tests

De nombreux outils existent pour aider et améliorer l'exécution des tests et les activités de logs de tests. Voici des exemples de ces outils :

- Outils d'exécution de tests (p. ex., pour exécuter des tests de régression)
- Outils de couverture (p. ex., couverture des exigences, couverture du code (D))
- Harnais de test (D)
- Framework de tests unitaires (D)

Outils pour la mesure de la performance et l'analyse dynamique

Les outils de mesure de performance et d'analyse dynamique sont essentiels pour assister les activités de test de performance et de montée en charge, car ces activités ne peuvent pas être effectuées manuellement. Voici des exemples de ces outils :

- Outils de test de performance
- Outils de monitoring
- Outils d'analyse dynamique (D)

Outils pour des besoins de test spécifiques

En plus des outils qui supportent le processus de test de manière générale, il existe de nombreux autres outils qui supportent des aspects de test plus spécifiques. Il s'agit par exemple d'outils qui se focalisent sur les éléments suivants :

- Évaluation de la qualité des données
- Conversion et migration des données
- Tests d'utilisabilité
- Tests d'accessibilité
- Tests de localisation
- Tests de sécurité
- Tests de portabilité (p. ex. tests de logiciels sur plusieurs plates-formes prises en charge)

6.1.2 Bénéfices et risques de l'automatisation des tests

Le simple fait d'acquérir un outil ne garantit pas le succès. Chaque nouvel outil introduit dans une organisation nécessitera des efforts pour obtenir des résultats tangibles et durables. L'utilisation d'outils de test offre des bénéfices et des opportunités potentielles, mais il y a aussi des risques. C'est particulièrement vrai pour les outils d'exécution des tests (ce que l'on appelle souvent les outils d'automatisation des tests).

Les bénéfices potentiels de l'utilisation d'outils d'aide à l'exécution des tests incluent :

- Réduction du travail manuel répétitif (p. ex., exécution de tests de régression, tâches de configuration/retrait de l'environnement, saisie des mêmes données de test et vérification par rapport aux normes de codage), ce qui permet de gagner du temps
- Plus grande cohérence et répétabilité (p. ex. les données de test sont créées de manière cohérente, les tests sont exécutés par un outil dans le même ordre avec la même fréquence et les tests sont rigoureusement dérivés des exigences)
- Évaluation plus objective (p. ex., mesures statiques, couverture)
- Accès facilité à l'information sur les tests (p. ex., statistiques et graphiques sur l'avancement des tests, les taux de défauts et les performances)

Les risques potentiels de l'utilisation d'outils en support aux tests incluent :

- Les attentes à l'égard de l'outil peuvent être irréalistes (y compris sur le plan des fonctionnalités et de la facilité d'utilisation)
- Le temps, le coût et l'effort pour l'introduction initiale d'un outil peuvent être sous-estimés (y compris pour la formation et l'expertise externe)

- Le temps et les efforts nécessaires pour obtenir des bénéfices significatifs et continus de l'outil peuvent être sous-estimés (y compris la nécessité de modifier le processus de test et d'améliorer continuellement la façon dont l'outil est utilisé)
- Les efforts nécessaires pour maintenir les actifs de test produits avec l'outil peuvent être sous-estimés
- Il est possible que l'on se repose trop sur l'outil (perçu comme un substitut à la conception ou à l'exécution des tests, ou l'utilisation de tests automatisés lorsque les tests manuels seraient préférables)
- Le contrôle des versions des actifs de test peut être négligé
- Les relations et les questions d'interopérabilité entre les outils principaux peuvent être ignorées, comme par exemple entre les outils de gestion des exigences, les outils de gestion de configuration, les outils de gestion des défauts et les outils provenant de plusieurs fournisseurs
- Le fournisseur de l'outil peut cesser ses activités, retirer l'outil ou vendre l'outil à un autre fournisseur
- Le fournisseur peut apporter une réponse médiocre pour le support, les mises à niveau et les corrections de défauts
- Un projet open source peut être interrompu
- Une nouvelle plate-forme ou technologie peut ne pas être prise en charge par l'outil
- Il se peut qu'il n'y ait pas de propriété claire de l'outil (par exemple, pour le conseil et les mises à jour)

6.1.3 Considérations particulières pour les outils d'exécution des tests et de gestion des tests

Afin d'assurer une mise en œuvre efficace et réussie, des aspects spécifiques doivent être pris en compte lors du choix et de l'intégration des outils d'exécution des tests et de gestion des tests dans une organisation.

Outils d'exécution des tests

Les outils d'exécution des tests exécutent des objets de test à l'aide de scripts de test automatisés. Ce type d'outil nécessite souvent des efforts importants pour obtenir des bénéfices significatifs.

Capturer les tests en enregistrant les actions d'un testeur manuel semble attrayant, mais cette approche n'est pas adaptée pour un grand nombre de scripts de test. Un script capturé est une représentation linéaire avec des données et des actions spécifiques pour chaque script. Ce type de script peut être instable lorsque des événements inattendus se produisent. La dernière génération de ces outils, qui tire parti de la technologie de capture d'images "intelligente", a augmenté l'utilité de cette classe d'outils, bien que les scripts générés nécessitent toujours une maintenance continue à mesure que l'interface utilisateur du système évolue au fil du temps.

Une approche de test piloté par les données sépare les entrées des tests et les résultats attendus, généralement sous la forme d'une feuille de calcul, et utilise un script de test plus générique qui peut lire les données d'entrée et exécuter le même script de test avec des données différentes. Les testeurs qui ne sont pas familiers avec le langage de script peuvent ainsi créer de nouvelles données de test pour ces scripts prédéfinis.

Dans une approche de test par mots-clés, un script générique traite des mots-clés décrivant les actions à entreprendre (également appelés mots-clés), qui font ensuite appel à des scripts d'implémentation des mots-clés pour traiter les données de test associées. Les testeurs (même s'ils ne maîtrisent pas le langage de script) peuvent alors définir des tests en utilisant les mots-clés et les données associées, qui peuvent être adaptés à l'application testée. D'autres détails et exemples d'approches de tests guidés par les données et les mots-clés sont donnés dans le syllabus CFTL/ISTQB de niveau avancé Automatisation de tests, Fewster 1999 et Buwalda 2001.

Les approches ci-dessus exigent que quelqu'un possède une expertise dans le langage de script (testeurs, développeurs ou spécialistes de l'automatisation des tests). Quelle que soit la technique de script utilisée, les résultats attendus pour chaque test doivent être comparés aux résultats effectifs du test, soit dynamiquement (pendant l'exécution du test), soit enregistrés pour une comparaison ultérieure (post-exécution).

Les outils de Model-Based Testing (MBT) permettent de capturer une spécification fonctionnelle sous la forme d'un modèle, comme par exemple un diagramme d'activité. Cette tâche est généralement exécutée par un concepteur de système. L'outil MBT interprète le modèle afin de créer des spécifications de cas de test qui peuvent ensuite être sauvegardées dans un outil de gestion des tests et/ou exécutées par un outil d'exécution de test (voir le syllabus de niveau fondation CFTL/ISTQB Model-Based Testing).

Outils de gestion des tests

Les outils de gestion des tests doivent souvent s'interfacer avec d'autres outils ou feuilles de calcul pour diverses raisons, telles que :

- Produire de l'information utile dans un format qui correspond aux besoins de l'organisation
- Maintenir une traçabilité cohérente des exigences dans un outil de gestion des exigences
- Etablir un lien avec les informations de version de l'objet de test dans l'outil de gestion de configuration

Cela est particulièrement important à considérer lorsqu'on utilise un outil intégré (p. ex. un outil de gestion du cycle de vie des applications), qui comprend un module de gestion des tests (et éventuellement un système de gestion des défauts), ainsi que d'autres modules (p. ex. l'information sur le calendrier et le budget du projet) utilisés par différents groupes au sein d'une organisation.

6.2 Utilisation efficace des outils

6.2.1 Principes de base pour la sélection des outils

Les principaux facteurs dont il faut tenir compte dans la sélection d'un outil pour une organisation sont les suivants :

- Évaluation de la maturité de l'organisation, de ses forces et de ses faiblesses
- Identification des opportunités pour un processus de test amélioré soutenu par des outils
- Compréhension des technologies utilisées par l'(s) objet(s) de test, afin de sélectionner un outil compatible avec cette technologie
- Les outils de build et d'intégration continue déjà utilisés au sein de l'organisation, afin d'assurer la compatibilité et l'intégration des outils
- Évaluation de l'outil par rapport à des exigences précises et à des critères objectifs

- Vérifier si l'outil est disponible pour une période d'essai gratuit (et pour combien de temps)
- Évaluation du fournisseur (y compris la formation, le support et les aspects commerciaux) ou du support pour les outils non commerciaux (p. ex., open source)
- Identification des besoins internes pour le coaching et le soutien dans l'utilisation de l'outil
- Évaluation des besoins de formation, en tenant compte des compétences en matière de test (et d'automatisation des tests) de ceux qui travailleront directement avec le ou les outil(s)
- Prise en compte des avantages et des inconvénients des différents modèles de licence (p. ex., commercial ou open source)
- Estimation d'un rapport coût-bénéfice basé sur une analyse de rentabilité concrète (si nécessaire)

Comme dernière étape, une preuve de concept (POC – Proof-of-Concept) devrait être effectuée pour déterminer si l'outil fonctionne efficacement avec le logiciel à tester et au sein de l'infrastructure actuelle ou, si nécessaire, pour identifier les changements nécessaires à cette infrastructure pour utiliser efficacement l'outil.

6.2.2 Projets pilotes pour l'introduction d'un outil dans une organisation

Après la sélection de l'outil et une preuve de concept réussie, l'introduction de l'outil sélectionné dans une organisation commence généralement par un projet pilote, dont les objectifs sont les suivants :

- Acquérir une connaissance approfondie de l'outil, comprendre ses forces et ses faiblesses
- Évaluer la façon dont l'outil s'intègre aux processus et pratiques existants et déterminer ce qu'il faudrait changer
- Décider des méthodes standard d'utilisation, de gestion, de sauvegarde et de maintenance de l'outil et des actifs de test (par exemple, décider des conventions de nommage des fichiers et des tests, sélectionner les normes de codage, créer des bibliothèques et définir la modularité des suites de test)
- Évaluer si les bénéfices seront obtenus à un coût raisonnable
- Comprendre les métriques que vous souhaitez voir recueillies et rapportées par l'outil, et configurer l'outil pour s'assurer que ces métriques peuvent être recueillies et rapportées

6.2.3 Facteurs de succès pour les outils

Les facteurs de succès pour l'évaluation, la mise en œuvre, le déploiement et le soutien continu à l'outillage au sein d'une organisation sont notamment les suivants :

- Déployer l'outil au reste de l'organisation de façon incrémentale
- Adapter et améliorer les processus en fonction de l'utilisation de l'outil
- Apporter de la formation, de l'encadrement et du conseil aux utilisateurs d'outils
- Définir des recommandations d'usage de l'outil (p. ex. normes internes pour l'automatisation)
- Mettre en place un moyen de collecte d'informations sur l'utilisation réelle de l'outil
- Suivre l'utilisation de l'outil et ses bénéfices
- Accompagner les utilisateurs d'un outil donné

- Recueillir les leçons tirées de l'expérience de tous les utilisateurs

Il est également important de s'assurer que l'outil est intégré au niveau technique et organisationnel dans le cycle de vie du développement du logiciel, ce qui peut impliquer des organisations différentes, responsables de l'exploitation et/ou des fournisseurs tiers.

Voir Graham 2012 pour des expériences et des conseils sur l'utilisation des outils d'exécution des tests.

7 Références

Normes

ISO/IEC/IEEE 29119-1 (2013) Software and systems engineering - Software testing - Part 1: Concepts and definitions

ISO/IEC/IEEE 29119-2 (2013) Software and systems engineering - Software testing - Part 2: Test processes

ISO/IEC/IEEE 29119-3 (2013) Software and systems engineering - Software testing - Part 3: Test documentation

ISO/IEC/IEEE 29119-4 (2015) Software and systems engineering - Software testing - Part 4: Test techniques

ISO/IEC 25010, (2011) Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) System and software quality models

ISO/IEC 20246: (2017) Software and systems engineering — Work product reviews

UML 2.5, Unified Modeling Language Reference Manual, http://www.omg.org/spec/UML/2.5.1/, 2017

Documents CFTL/ISTQB

Glossaire CFTL/ISTQB au niveau fondation

Présentation générale de la certification CFTL/ISTQB au niveau fondation en version 2018

CFTL/ISTQB-MBT Niveau Fondation – Syllabus Model-Based Testing

CFTL/ISTQB-AT Niveau Fondation – Syllabus Testeur Agile

CFTL/ISTQB-ATA Niveau Avancé – Syllabus Analyste de Test

CFTL/ISTQB-ATA Niveau Avancé – Syllabus Analyste Technique de Test

CFTL/ISTQB-ATM Niveau Avancé – Syllabus Test Manager

CFTL/ISTQB-SEC Niveau Avancé – Syllabus Testeur de sécurité

CFTL/ISTQB-TAE Niveau Avancé – Syllabus Automaticien de Test

CFTL/ISTQB-ETM Niveau Expert – Syllabus Gestion des tests

CFTL/ISTQB-EITP Niveau Expert – Syllabus Améliorer le processus de test

Livres et articles

Beizer, B. (1990*) Software Testing Techniques (2e)*, Van Nostrand Reinhold: Boston MA

Black, R. (2017) *Agile Testing Foundations*, BCS Learning & Development Ltd: Swindon UK

Black, R. (2009) *Managing the Testing Process (3e)*, John Wiley & Sons: New York NY

Buwalda, H. et al. (2001) *Integrated Test Design and Automation*, Addison Wesley: Reading MA

Copeland, L. (2004) *A Practitioner's Guide to Software Test Design*, Artech House: Norwood MA

Craig, R. and Jaskiel, S. (2002) *Systematic Software Testing*, Artech House: Norwood MA

Crispin, L. and Gregory, J. (2008) *Agile Testing*, Pearson Education: Boston MA

Fewster, M. and Graham, D. (1999) *Software Test Automation*, Addison Wesley: Harlow UK

Gilb, T. and Graham, D. (1993) *Software Inspection*, Addison Wesley: Reading MA

Graham, D. and Fewster, M. (2012) *Experiences of Test Automation*, Pearson Education: Boston MA

Gregory, J. and Crispin, L. (2015) *More Agile Testing*, Pearson Education: Boston MA

Jorgensen, P. (2014) *Software Testing, A Craftsman's Approach (4e)*, CRC Press: Boca Raton FL

Kaner, C., Bach, J. and Pettichord, B. (2002) *Lessons Learned in Software Testing*, John Wiley & Sons: New York NY

Kaner, C., Padmanabhan, S. and Hoffman, D. (2013) *The Domain Testing Workbook*, Context-Driven Press: New York NY

Kramer, A., Legeard, B. (2016) *Model-Based Testing Essentials: Guide to the ISTQB Certified Model-Based Tester: Foundation Level*, John Wiley & Sons: New York NY

Myers, G. (2011) *The Art of Software Testing*, (3e), John Wiley & Sons: New York NY

Sauer, C. (2000) "The Effectiveness of Software Development Technical Reviews: A Behaviorally Motivated Program of Research," *IEEE Transactions on Software Engineering*, *Volume 26, Issue 1, pp 1-*

Shull, F., Rus, I., Basili, V. July 2000. "How Perspective-Based Reading can Improve Requirement Inspections." *IEEE Computer, Volume 33, Issue 7, pp 73-79*

van Veenendaal, E. (ed.) (2004) *The Testing Practitioner* (Chapters 8 - 10), UTN Publishers: The Netherlands

Wiegers, K. (2002) *Peer Reviews in Software*, Pearson Education: Boston MA

Weinberg, G. (2008) *Perfect Software and Other Illusions about Testing*, Dorset House: New York NY

Autres ressources (non référencées directement dans ce syllabus)

Black, R., van Veenendaal, E. and Graham, D. (2012) *Foundations of Software Testing: ISTQB Certification (3e)*, Cengage Learning: London UK

Hetzel, W. (1993) *Complete Guide to Software Testing (2e)*, QED Information Sciences: Wellesley MA

Spillner, A., Linz, T., and Schaefer, H. (2014) *Software Testing Foundations (4e)*, Rocky Nook: San Rafael CA

8 Annexe A - Contexte d'élaboration du Syllabus

Historique de ce document

Ce document est le Syllabus CFTL/ISTQB Testeur certifié au niveau fondation, la qualification internationale au premier niveau approuvée par l'ISTQB (www.istqb.org).

Ce document a été préparé entre 2014 et 2018 par un groupe de travail composé de membres nommés par l'International Software Testing Qualifications Board (ISTQB). La version 2018 a d'abord été revue par des représentants de tous les comités membres de l'ISTQB, puis par des représentants de la communauté internationale des tests logiciels.

Objectifs de la certification au niveau fondation

- Faire reconnaître les tests comme une spécialisation essentielle et professionnelle en génie logiciel
- Fournir un cadre standard pour le déroulement de carrière des testeurs
- Permettre aux testeurs dûment qualifiés d'être reconnus par les employeurs, les clients et les pairs, et de renforcer le professionnalisme des testeurs
- Promouvoir des pratiques de test cohérentes et efficaces dans toutes les disciplines du génie logiciel
- Identifier les sujets de test qui sont pertinents et de valeur pour l'industrie
- Permettre aux éditeurs de logiciels d'engager des testeurs certifiés et d'obtenir ainsi un avantage commercial par rapport à leurs concurrents en mettant en avant leur politique de recrutement de testeurs
- Offrir la possibilité aux testeurs et à ceux qui s'intéressent aux tests d'acquérir une qualification reconnue à l'échelle internationale dans le domaine

Objectifs de la qualification internationale

- Pouvoir comparer les connaissances en matière de tests dans différents pays
- Permettre aux testeurs de dépasser plus facilement les frontières nationales
- Permettre aux projets multinationaux/internationaux d'avoir une compréhension commune des questions relatives aux tests
- Augmenter le nombre de testeurs qualifiés dans le monde entier
- Avoir plus d'impact/valeur en tant qu'initiative internationale qu'une approche spécifique à un pays
- Développer un corpus international commun de compréhension et de connaissances sur les tests par le biais du syllabus et de la terminologie, et augmenter le niveau de connaissances sur les tests pour tous les participants
- Promouvoir le test en tant que profession dans un plus grand nombre de pays
- Permettre aux testeurs d'obtenir une qualification reconnue dans leur langue maternelle
- Permettre le partage des connaissances et des ressources documentaires entre les pays

- Assurer la reconnaissance internationale des testeurs et de cette qualification grâce à la participation de nombreux pays

Conditions d'admission pour cette qualification

Le critère d'admission à l'examen de certification CFTL/ISTQB Testeur Certifié au niveau Fondation est que les candidats s'intéressent aux tests logiciels. Cependant, il est aussi fortement recommandé que chaque candidat :

- possède un minimum d'expérience dans le développement ou le test de logiciels, par exemple six mois d'expérience en tant que testeur de système ou d'acceptation par l'utilisateur ou en tant que développeur de logiciels
- suive un cours qui a été accrédité par l'un des comités membres reconnus par l'ISTQB selon les normes de l'ISTQB.

Contexte et historique du certificat de niveau fondation en tests logiciels

La certification indépendante des testeurs de logiciels a commencé au Royaume-Uni avec l'Information Systems Examination Board (ISEB) de la British Computer Society, lorsqu'un Software Testing Board a été créé en 1998 (www.bcs.org.uk/iseb). En 2002, l'ASQF en Allemagne a commencé à soutenir un programme de qualification des testeurs allemands (www.asqf.de). Le présent syllabus est basé sur les syllabus de l'ISEB et de l'ASQF ; il comprend un contenu réorganisé, mis à jour et additionnel, et l'accent est mis sur les sujets qui apporteront la plus grande aide pratique aux testeurs.

Un certificat de niveau fondation existant en tests de logiciels (par exemple, de l'ISEB, de l'ASQF ou d'un comité membre reconnu par l'ISTQB) délivré avant la délivrance de ce certificat international sera considéré comme équivalent au certificat international. Le certificat de niveau fondation ne prend pas fin et n'a pas besoin d'être renouvelé. La date de son attribution est indiquée sur le certificat.

Dans chaque pays participant, les aspects locaux sont contrôlés par un comité national ou régional reconnu par l'ISTQB. Les attributions des comités membres sont spécifiées par l'ISTQB, mais sont mises en œuvre dans chaque pays. Les responsabilités des comités nationaux comprennent l'accréditation des prestataires de formation et la mise en place des examens.

9 Annexe B - Objectifs d'apprentissage / Niveau cognitif des connaissances

Les types d'objectifs d'apprentissage suivants sont définis comme s'appliquant à ce syllabus. Chaque sujet du syllabus sera examiné en fonction de l'objectif d'apprentissage qui lui est assigné.

Niveau 1 : Se souvenir (K1)

Le candidat reconnaîtra, se rappellera et se souviendra des termes ou des concepts.

Mots clés : Identifier, se souvenir, retrouver, rappeler, reconnaître, savoir

Exemples :

Peut reconnaître la définition de “défaillance” comme :

- “non livraison d’un service à l’utilisateur final ou tout autre personne impliquée” ou
- “Déviation constatée du composant ou système de la fourniture, service ou résultat attendu”

Niveau 2 : comprendre (K2)

Le candidat peut sélectionner les raisons ou explications pour des affirmations liées au sujet traité, et peut résumer, différencier, classer et donner des exemples sur les concepts de test

Mots clés : résumer, généraliser, abstraire, classer, comparer, cartographier, différencier, illustrer, interpréter, traduire, représenter, déduire, conclure, catégoriser, construire des modèles

Exemples :

Peut expliquer les raisons pour lesquelles les tests doivent être exécutés aussi tôt que possible :

- Pour trouver des défauts quand il est intéressant de les supprimer.
- Pour trouver d’abord les défauts les plus importants.

Peut expliquer les similitudes et les différences entre les tests d’intégration et les tests système :

- Similitudes : tester plus d’un composant, et tester des aspects non-fonctionnels.
- Différences : les tests d’intégration se concentrent sur les interfaces et les interactions, les tests système se focalisent sur les aspects de systèmes entiers, tels le processus de bout en bout.

Niveau 3 : Appliquer (K3)

Le candidat peut sélectionner l’application correcte d’un concept ou d’une technique et l’appliquer à un contexte donné.

Mots clés : Implémenter, exécuter, utiliser, suivre une procédure, appliquer une procédure

Exemples :

- Peut identifier les valeurs limites pour des partitions valides et invalides
- Peut sélectionner les cas de test nécessaires à la couverture de toutes les transitions pour un diagramme d’état donné

Références

(pour les niveaux cognitifs des objectifs d'apprentissage)

Anderson, L. W. and Krathwohl, D. R. (eds) (2001) *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*, Allyn & Bacon

10 Annexe C - Notes de la version

Le syllabus CFTL/ISTQB au niveau Fondation en version 2018 est une mise à jour majeure et une réécriture de la version 2011. Pour cette raison, il n'y a pas de notes de version détaillées par chapitre et par section. Cependant, un résumé des principaux changements est fourni ici. De plus, dans un document séparé, l'ISTQB assure la traçabilité entre les objectifs d'apprentissage de la version 2011 du Syllabus au niveau Fondation et les objectifs d'apprentissage de la version 2018 du Syllabus au niveau Fondation, en indiquant ceux qui ont été ajoutés, mis à jour ou supprimés.

Au début de 2017, plus de 550 000 personnes dans plus de 100 pays ont passé l'examen du niveau fondation et plus de 500 000 sont des testeurs certifiés dans le monde entier. Dans la mesure où ils ont tous lu le Syllabus de niveau Fondation pour pouvoir passer l'examen, cela fait que ce syllabus est probablement le document sur les tests logiciels le plus lu de tous les temps !

Cette mise à jour majeure est faite dans le respect de ce patrimoine et pour renforcer la valeur que l'ISTQB apporte aux 500 000 prochaines personnes dans la communauté mondiale des tests.

Dans cette version, tous les objectifs d'apprentissage ont été modifiés pour les rendre atomiques et pour créer une traçabilité précise de chaque objectif d'apprentissage à la ou aux sections de contenu (et aux questions d'examen) qui sont liées à cet objectif d'apprentissage, et pour avoir une traçabilité précise de la ou des sections de contenu (et des questions d'examen) jusqu'à l'objectif d'apprentissage associé. De plus, les répartitions de temps des chapitres ont été rendues plus réalistes que celles de la version 2011 du syllabus, en utilisant des heuristiques et des formules éprouvées utilisées avec d'autres syllabus de l'ISTQB, qui sont basées sur une analyse des objectifs d'apprentissage à couvrir dans chaque chapitre.

Bien qu'il s'agisse d'un syllabus de niveau Fondation, exprimant les meilleures pratiques et techniques qui ont fait leurs preuves au fil du temps, nous avons apporté des changements pour moderniser la présentation du matériel, notamment en termes de méthodes de développement de logiciels (par exemple, Scrum et déploiement continu) et de technologies (par exemple, l'Internet des objets). Nous avons mis à jour les normes de référence pour les rendre plus récentes comme suit :

1. ISO/IEC/IEEE 29119 remplace IEEE Standard 829.
2. ISO/IEC 25010 remplace ISO 9126.
3. ISO/IEC 20246 remplace IEEE 1028.

En outre, puisque le portefeuille des certifications de l'ISTQB s'est considérablement élargi au cours de la dernière décennie, nous avons ajouté des renvois à des documents connexes d'autres syllabus de l'ISTQB, le cas échéant, ainsi qu'un examen minutieux de l'alignement avec tous les syllabus et avec le glossaire de l'ISTQB. L'objectif est de rendre cette version plus facile à lire, à comprendre, à apprendre et à traduire, en mettant l'accent sur une plus grande utilité pratique et sur l'équilibre entre les connaissances et les compétences.

Pour une analyse détaillée des modifications apportées dans cette version, reportez-vous au document de présentation générale de la certification CFTL/ISTQB au niveau fondation en version 2018.