Testeur Certifié

Syllabus Niveau Fondation

v4.0

International Software Testing Qualifications Board

Comité Français des Tests Logiciels

Avis de copyright

- La traduction française est la propriété du CFTL. Elle a été réalisée par un groupe d'experts en tests logiciels : Eric Riou du Cosquer, Olivier Denoo et Bruno Legeard.

Historique des modifications

Version	Date	Remarques
CTFL v4.0	12 avril 2023	CTFL v4.0 – Version majeure générale
CFTL 2018FR	31 octobre 2018	Version française mise à jour avec la correction d'erreurs de typographie
CFTL 2018FR	31 août 2018	Version française avec corrections mineures de traduction
CFTL 2018FR	30 juin 2018	Version française
ISTQB 2018	27avril 2018	Version finale
ISTQB 2018	12 février 2018	Version Beta
ISTQB 2018	19 janvier 2018	Version Alpha 3.0
ISTQB 2018	15 janvier 2018	Version Alpha 2.9 - Revue interne (cross review) et édition technique
ISTQB 2018	9 décembre 2017	Version Alpha 2.5
ISTQB 2018	22 novembre 2017	Version Alpha 2.0
ISTQB 2018	12 juin 2017	Version Alpha
CFTL 2011FR	3 novembre 2011	Testeur Certifié Niveau Fondation Mise à jour – voir **Erreur ! Source du renvoi introuvable.**1
CFTL 2010FR	1 septembre 2010	Testeur Certifié Niveau Fondation Mise à jour – voir **Erreur ! Source du renvoi introuvable.**
ISTQB 2010	30 mars 2010	Certified Tester Foundation Level Syllabus Maintenance Release – voir **Erreur ! Source du renvoi introuvable.**
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	25février 1999	ISEB Software Testing Foundation Syllabus V2.0 25 February 1999

Table des matières

Remerciements

Ce document a été officiellement livré par l'assemblée générale de l'ISTQB® le 21 avril 2023.

Il a été produit par une équipe des groupes de travail conjoints de l'ISTQB® sur le Niveau Fondation et Agile : Laura Albert, Renzo Cerquozzi (vice-président), Wim Decoutere, Klaudia Dussa-Zieger, Chintaka Indikadahena, Arnika Hryszko, Martin Klonk, Kenji Onishi, Michaël Pilaeten (co-responsable), Meile Posthuma, Gandhinee Rajkomar, Stuart Reid, Eric Riou du Cosquer (responsable), Jean-François Riverin, Adam Roman, Lucjan Stapp, Stephanie Ulrich (co-responsable), Eshraka Zakaria.

L'équipe remercie Stuart Reid, Patricia McQuaid et Leanne Howard pour leur revue technique, ainsi que l'équipe de réviseurs et les membres des conseils d'administration pour leurs suggestions et leur contribution.

Les personnes suivantes ont participé à la revue, aux commentaires et au vote de ce syllabus : Adam Roman, Adam Scierski, Ágota Horváth, Ainsley Rood, Ale Rebon Portillo, Alessandro Collino, Alexander Alexandrov, Amanda Logue, Ana Ochoa, André Baumann, André Verschelling, Andreas Spillner, Anna Miazek, Armin Born, Arnd Pehl, Arne Becher, Attila Gyúri, Attila Kovács, Beata Karpinska, Benjamin Timmermans, Blair Mo, Carsten Weise, Chinthaka Indikadahena, Chris Van Bael, Ciaran O'Leary, Claude Zhang, Cristina Sobrero, Dandan Zheng, Dani Almog, Daniel Säther, Daniel van der Zwan, Danilo Magli, Darvay Tamás Béla, Dawn Haynes, Dena Pauletti, Dénes Medzihradszky, Doris Dötzer, Dot Graham, Edward Weller, Erhardt Wunderlich, Eric Riou du Cosquer, Florian Fieber, Fran O'Hara, François Vaillancourt, Frans Dijkman, Gabriele Haller, Gary Mogyorodi, Georg Sehl, Géza Bujdosó, Giancarlo Tomasig, Giorgio Pisani, Gustavo Márquez Sosa, Helmut Pichler, Hongbao Zhai, Horst Pohlmann, Ignacio Trejos, Ilia Kulakov, Ine Lutterman, Ingvar Nordström, Iosif Itkin, Jamie Mitchell, Jan Giesen, Jean-François Riverin, Joanna Kazun, Joanne Tremblay, Joëlle Genois, Johan Klintin, John Kurowski, Jörn Münzel, Judy McKay, Jürgen Beniermann, Karol Frühauf, Katalin Balla, Kevin Kooh, Klaudia Dussa-Zieger, Klaus Erlenbach, Klaus Olsen, Krisztián Miskó, Laura Albert, Liang Ren, Lijuan Wang, Lloyd Roden, Lucjan Stapp, Mahmoud Khalaili, Marek Majernik, Maria Clara Choucair, Mark Rutz, Markus Niehammer, Martin Klonk, Márton Siska, Matthew Gregg, Matthias Hamburg, Mattijs Kemmink, Maud Schlich, May Abu-Sbeit, Meile Posthuma, Mette Bruhn-Pedersen, Michal Tal, Michel Boies, Mike Smith, Miroslav Renda, Mohsen Ekssir, Monika Stocklein Olsen, Murian Song, Nicola De Rosa, Nikita Kalyani, Nishan Portoyan, Nitzan Goldenberg, Ole Chr. Hansen, Patricia McQuaid, Patricia Osorio, Paul Weymouth, Pawel Kwasik, Peter Zimmerer, Petr Neugebauer, Piet de Roo, Radoslaw Smilgin, Ralf Bongard, Ralf Reissing, Randall Rice, Rik Marselis, Rogier Ammerlaan, Sabine Gschwandtner, Sabine Uhde, Salinda Wickramasinghe, Salvatore Reale, Sammy Kolluru, Samuel Ouko, Stephanie Ulrich, Stuart Reid, Surabhi Bellani, Szilard Szell, Tamás Gergely, Tamás Horváth, Tatiana Sergeeva, Tauhida Parveen, Thaer Mustafa, Thomas Eisbrenner, Thomas Harms, Thomas Heller, Tobias Letzkus, Tomas Rosenqvist, Werner Lieblang, Yaron Tsubery, Zhenlei Zuo et Zsolt Hargitai.

Groupe de travail de l'ISTQB Niveau Fondation (édition 2018) : Klaus Olsen (responsable), Tauhida Parveen (vice-présidente), Rex Black (manager du projet), Eshraka Zakaria, Debra Friedenberg, Ebbe Munk, Hans Schaefer, Judy McKay, Marie Walsh, Meile Posthuma, Mike Smith, Radoslaw Smilgin, Stephanie Ulrich, Steve Toms, Corne Kruger, Dani Almog, Eric Riou du Cosquer, Igal Levi, Johan Klintin, Kenji Onishi, Rashed Karim, Stevan Zivanovic, Sunny Kwon, Thomas Müller, Vipul Kocher, Yaron Tsubery 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 comme suit :

1. aux Membres de l'ISTQB, afin de traduire dans leur langue locale et d'accréditer les fournisseurs de formation. Les Membres peuvent adapter ce syllabus à leurs besoins linguistiques particuliers et ajouter des références pour l'adapter à leurs publications locales.
2. aux organismes de certification, afin de produire les questions d'examen dans leur langue locale en fonction des objectifs d'apprentissage pour ce syllabus.
3. aux organismes de formation, afin de produire le matériel de formation et déterminer les méthodes pédagogiques appropriées.
4. aux candidats à la certification, pour se préparer à l'examen de certification (dans le cadre d'un cours de formation ou indépendamment).
5. à 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.

0.2. Le niveau Fondation pour les testeurs de logiciels certifiés

La certification de niveau Fondation s'adresse à toute personne impliquée dans les tests de logiciels. Il s'agit notamment des testeurs, des analystes de test, des ingénieurs de test, des consultants en test, des test managers, des développeurs de logiciels et des membres de l'équipe de développement. Cette qualification de niveau Fondation convient également à toute personne souhaitant avoir une compréhension de base des tests de logiciels, comme les chefs de projets, les responsables qualité, les Product Owners, les responsables du développement de logiciels, les analystes métier, les directeurs informatiques et les consultants en management. Les titulaires du certificat de niveau Fondation pourront accéder à des certifications de niveau supérieur en matière de tests de logiciels.

0.3. Parcours de carrière pour les testeurs

Le programme de l'ISTQB® apporte un soutien aux professionnels du test à tous les stades de leur carrière, en leur offrant des connaissances à la fois étendues et approfondies. Les personnes ayant obtenu la certification ISTQB® de niveau Fondation peuvent également être intéressées par les niveaux Avancés principaux (Analyste de Test, Analyste Technique de Test et Test Manager) et par la suite le niveau Expert (Management des tests ou Amélioration du processus de test). Toute personne cherchant à développer ses compétences en matière de pratiques de test dans un environnement Agile pourrait envisager les certifications Testeur Technique Agile ou "Agile Test Leadership at Scale". La filière Spécialiste offre une connaissance approfondie des domaines qui ont des approches de test et des

activités de test spécifiques (par exemple, dans l'automatisation des tests, les tests d'IA, les tests basés sur des modèles, les tests d'applications mobiles) ; qui sont liés à des domaines de test spécifiques (par exemple, les tests de performance, les tests d'utilisabilité, les tests d'acceptation, les tests de sécurité) ; ou qui regroupent le savoir-faire en matière de tests pour certains domaines industriels (par exemple, l'automobile ou les jeux). Pour obtenir les dernières informations sur le programme de certification des testeurs de l'ISTQB® et du CFTL, veuillez consulter les sites www.istqb.org et www.cftl.fr.

0.4. Objectifs métiers

Cette section liste les 14 objectifs métiers attendus d'une personne ayant obtenu la certification de Niveau Fondation.

Un testeur certifié Niveau Fondation peut …

FL-BO1	Comprendre ce qu'est le test et pourquoi il est bénéfique
FL-BO2	Comprendre les concepts fondamentaux du test logiciel
FL-BO3	Identifier l'approche et les activités de test à mettre en œuvre en fonction du contexte du test
FL-BO4	Évaluer et améliorer la qualité de la documentation
FL-BO5	Accroître l'efficacité et l'efficience des tests
FL-BO6	Aligner le processus de test sur le cycle de vie du développement logiciel
FL-BO7	Comprendre les principes de la gestion des tests
FL-BO8	Rédiger et communiquer des rapports de défauts clairs et compréhensibles
FL-BO9	Comprendre les facteurs qui influencent les priorités et les efforts liés aux tests
FL-BO10	Travailler au sein d'une équipe interfonctionnelle
FL-BO11	Connaître les risques et les bénéfices liés à l'automatisation des tests
FL-BO12	Identifier les compétences essentielles requises pour le test
FL-BO13	Comprendre l'impact des risques sur les tests
FL-BO14	Rendre compte efficacement de l'état d'avancement et de la qualité des tests

0.5. Objectifs d'apprentissage examinables et niveaux de connaissance

Les objectifs d'apprentissage soutiennent les objectifs métier et sont utilisés pour créer les examens Testeur Certifié de niveau Fondation. En général, tous les contenus des chapitres 1 à 6 de ce syllabus sont examinables au niveau K1. C'est-à-dire qu'il peut être demandé au candidat de reconnaître, de se souvenir ou de rappeler un mot-clé ou un concept mentionné dans l'un des six chapitres. Les niveaux spécifiques des objectifs d'apprentissage sont indiqués au début de chaque chapitre et classés comme suit:

- K1: se souvenir

- K2: comprendre
- K3: appliquer

Des détails supplémentaires et des exemples d'objectifs d'apprentissage sont donnés dans l'annexe A. Tous les termes listés comme mots-clés, juste en dessous des titres des chapitres, doivent être mémorisés (K1), même s'ils ne sont pas explicitement mentionnés dans les objectifs d'apprentissage.

0.6. L'examen de certification de niveau Fondation

L'examen de certification du niveau Fondation est basé sur ce syllabus. Les réponses aux questions de l'examen peuvent nécessiter l'utilisation d'éléments basés sur plus d'une section de ce syllabus. Toutes les sections du syllabus sont examinables, à l'exception de l'introduction et des annexes. Le contenu des standards et des livres inclus comme référence (chapitre 7), n'est pas examinable au-delà des extraits résumés dans le syllabus lui-même. Se référer au document Structures et règles de l'examen de Niveau Fondation.

0.7. Accréditation

Un Membre de l'ISTQB® peut accréditer des organismes de formation dont le matériel de formation suit ce syllabus. Les organismes de formation doivent obtenir les directives d'accréditation auprès du Membre ou de l'organisme qui effectue l'accréditation. Un cours accrédité est reconnu comme étant conforme à ce syllabus et peut comporter un examen de l'ISTQB®. Les directives d'accréditation pour ce syllabus suivent les directives générales d'accréditation publiées par le groupe de travail sur la gestion des processus et la conformité.

0.8. Prise en compte des normes

Certaines normes sont référencées dans le syllabus de base (par exemple, les normes IEEE ou ISO). Ces références fournissent un cadre (comme les références à la norme ISO 25010 concernant les caractéristiques de qualité) ou une source d'informations supplémentaires si le lecteur le souhaite. Les documents de normalisation ne sont pas destinés à être examinés. Pour plus d'informations sur les normes, voir le chapitre 7.

0.9. Rester à jour

L'industrie du logiciel évolue rapidement. Pour faire face à ces changements et permettre aux parties prenantes d'accéder à des informations pertinentes et actuelles, les groupes de travail de l'ISTQB® ont créé des liens sur le site web www.istqb.org, qui renvoient à des documents complémentaires et à des modifications apportées aux normes. Ces informations ne sont pas examinables dans le cadre du syllabus de niveau Foundation.

0.10. Niveau de détail

Le niveau de détail de ce syllabus permet de proposer des formations et des examens cohérents à l'échelle internationale. Pour atteindre cet objectif, le syllabus se compose:

- Des objectifs généraux d'instruction, décrivant les intentions du niveau Fondation
- D'une liste de termes (mots-clés) que les candidats doivent être capables de mémoriser
- Des objectifs d'apprentissage pour chaque domaine de connaissance, décrivant les résultats d'apprentissage cognitifs à atteindre
- D'une description des concepts clés, y compris des références à des sources reconnues

Le contenu du syllabus n'est pas une description de l'ensemble du domaine de connaissance des tests de logiciels ; il reflète le niveau de détail à couvrir dans les cours de formation de niveau Fondation. Il se concentre sur les concepts et techniques de test qui peuvent être appliqués à tous les projets logiciels indépendamment du Cycle de Vie du Développement Logiciel employé.

0.11. Organisation du syllabus

Il y a six chapitres qui comportent un contenu pouvant être examiné. Le titre de niveau supérieur de chaque chapitre indique le temps de formation pour le chapitre. Le minutage n'est pas fourni en dessous du niveau du chapitre. Pour les formations accréditées, le syllabus exige un minimum de 18,75 heures (18 heures et 55 minutes) d'enseignement, réparties entre les six chapitres comme suit:

- Chapitre 1 : Fondamentaux des tests (180 minutes)
    - Le candidat apprend les principes de base sur les tests, les raisons pour lesquelles les tests sont requis et quels sont les objectifs des tests.
    - Le candidat comprend le processus de test, les principales activités de test et le testware.
    - Le candidat comprend les compétences essentielles pour les tests.
- Chapitre 2 : Tester tout au long du développement logiciel (130 minutes)
    - Le candidat apprend comment les tests sont intégrés dans les différentes approches de développement.
    - Le candidat apprend les concepts des approches pilotées par les tests, ainsi que DevOps.
    - Le candidat apprend les différents niveaux de test, les types de test et les tests de maintenance.
- Chapitre 3 : Test statique (80 minutes)
    - Le candidat apprend les bases du test statique, le processus de feedback et de revue.
- Chapitre 4 : Analyse et conception des tests (390 minutes)
    - Le candidat apprend à appliquer les techniques de test boîte noire, boîte blanche et les techniques de test basées sur l'expérience pour dériver des cas de test à partir de divers produits activités.
    - Le candidat apprend à connaître l'approche de test basée sur la collaboration..

- Chapitre 5 : Gestion des activités de test (335 minutes)
    - Le candidat apprend à planifier les tests en général et à estimer l'effort de test.
    - Le candidat apprend comment les risques peuvent influencer le périmètre des tests.
    - Le candidat apprend à piloter et contrôler les activités de test
    - Le candidat apprend comment la gestion de la configuration soutient les tests.
    - Le candidat apprend à rapporter les défauts d'une manière claire et compréhensible.
- Chapitre 6: Outils de test (20 minutes)
    - Le candidat apprend à classer les outils et à comprendre les risques et les bénéfices de l'automatisation des tests.

1. Fondamentaux des tests – 180 minutes

Mots-clés

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, implémentation des tests, pilotage des tests, objet de test, objectif de test, planification des tests, procédure de test, résultat de test, test, testware, validation, vérification.

Objectifs d'apprentissage pour le chapitre 1:

1.1 Qu'est-ce que le test ?

FL-1.1.1 (K1) Identifier les objectifs habituels du test

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

1.2 Pourquoi est-il nécessaire de tester?

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

FL-1.2.2 (K1) Rappeler la relation entre les tests et assurance qualité

FL-1.2.3 (K2) Faire la distinction entre la cause racine, l'erreur, le défaut et la défaillance.

1.3 Principes du test

FL-1.3.1 (K2) Expliquer les sept principes du test

1.4 Activités de test, testware et rôles dans le test

FL-1.4.1 (K2) Résumer les différentes activités et tâches de test

FL-1.4.2 (K2) Expliquer l'impact du contexte sur le processus de test

FL-1.4.3 (K2) Différencier les composants du testware qui soutiennent les activités de test

FL-1.4.4 (K2) Expliquer la valeur du maintien de la traçabilité

FL-1.4.5 (K2) Comparer les différents rôles dans le test

1.5 Compétences essentielles et bonnes pratiques en matière de test

FL-1.5.1 (K2) Donnez des exemples de compétences génériques requises pour le test

FL-1.5.2 (K1) Rappeler les avantages de l'approche équipe intégrée

FL-1.5.3 (K2) Distinguer les avantages et les inconvénients de l'indépendance du test

1.1. Qu'est-ce que le test?

Les systèmes logiciels font partie intégrante de notre vie quotidienne. La plupart des gens ont déjà eu affaire à un logiciel qui ne fonctionnait pas comme prévu. Un logiciel qui ne fonctionne pas correctement peut entraîner de nombreux problèmes, notamment une perte d'argent, de temps ou de réputation de l'entreprise et, dans des cas extrêmes, des blessures ou la mort. Le test de logiciels permet d'évaluer la qualité des logiciels et de réduire le risque de défaillance en cours d'utilisation.

Le test de logiciels est un ensemble d'activités visant à découvrir les défauts et à évaluer la qualité des artefacts logiciels. Ces artefacts, lorsqu'ils sont testés, sont connus sous le nom d'objets de test. Une idée fausse très répandue à propos du test (en tant qu'activité) est qu'il consiste uniquement à exécuter des tests (c'est-à-dire à faire fonctionner le logiciel et à vérifier les résultats des tests). Cependant, le test de logiciels comprend également d'autres activités et doit être aligné sur le cycle de vie du développement logiciel (voir chapitre 2).

Une autre idée fausse et très répandue à propos du test est qu'il se concentre entièrement sur la vérification de l'objet de test. Si le test implique la vérification, c'est-à-dire le contrôle de la conformité du système aux exigences spécifiées, il implique également la validation, c'est-à-dire le contrôle de la conformité du système aux besoins des utilisateurs et des autres parties prenantes dans son environnement opérationnel.

Le test peut être dynamique ou statique. Le test dynamique implique l'exécution du logiciel, ce qui n'est pas le cas du test statique. Le test statique comprend les revues (voir chapitre 3) et l'analyse statique. Le test dynamique utilise différents types de techniques de test et d'approches de test pour dériver des cas de test (voir chapitre 4).

Le test n'est pas seulement une activité technique. Il doit également être correctement planifié, géré, estimé, piloté et contrôlé (voir chapitre 5).

Les testeurs utilisent des outils (voir chapitre 6), mais il est important de rappeler que le test est en grande partie une activité intellectuelle, exigeant des testeurs qu'ils aient des connaissances spécialisées, qu'ils utilisent des compétences analytiques et qu'ils appliquent la pensée critique et la pensée systémique (Myers 2011, Roman 2018).

Le standard ISO/IEC/IEEE 29119-1 fournit de plus amples informations sur les concepts du test de logiciels.

1.1.1. Objectifs du test

Les objectifs typiques du test sont les suivants:

- Évaluer les produits d'activités tels que les exigences, les User Stories, les conceptions et le code.
- Provoquer des défaillances et trouver des défauts.
- Assurer la couverture requise d'un objet de test.
- Réduire le niveau de risque d'une qualité logicielle insuffisante.
- Vérifier si les exigences spécifiées ont été satisfaites.
- Vérifier qu'un objet de test est conforme aux exigences contractuelles, légales et réglementaires.

- Fournir des informations aux parties prenantes pour leur permettre de prendre des décisions éclairées.
- Construire la confiance dans la qualité de l'objet de test.
- Valider si l'objet de test est complet et fonctionne comme attendu par les parties prenantes.

Les objectifs du test peuvent varier en fonction du contexte, qui comprend le produit d'activités testé, le niveau de test, les risques, le cycle de vie du développement logiciel (SDLC) suivi et les facteurs liés au contexte métier, par exemple la structure de l'entreprise, les considérations concurrentielles ou le délai de mise sur le marché.

1.1.2. Test et débogage

Le test et le débogage sont des activités distinctes. Les tests peuvent déclencher des défaillances causées par des défauts dans le logiciel (test dynamique) ou constater directement des défauts dans l'objet de test (test statique).

Lorsque le test dynamique (voir chapitre 4) déclenche une défaillance, le débogage consiste à trouver les causes de cette défaillance (défauts), à analyser ces causes et à les éliminer. Dans ce cas, le processus de débogage typique implique les tâches suivantes:

- Reproduction d'une défaillance.
- Diagnostic (trouver la cause racine).
- Correction de la cause.

Un test de confirmation ultérieur permet de vérifier si les correctifs apportés ont résolu le problème. De préférence, le test de confirmation est effectué par la même personne que celle qui a réalisé le test initial. Des tests de régression ultérieurs peuvent également être effectués pour vérifier si les corrections entraînent des défaillances dans d'autres parties de l'objet test (voir la section 2.2.3 pour plus d'informations sur les tests de confirmation et les tests de régression).

Lorsque le test statique identifie un défaut, le débogage consiste à l'éliminer. Il n'est pas nécessaire de le reproduire ou de le diagnostiquer, puisque le test statique constate directement les défauts et ne peut pas provoquer de défaillances (voir chapitre 3).

1.2. Pourquoi est-il nécessaire de tester ?

Le test, en tant que forme de contrôle de la qualité, aide à atteindre les objectifs convenus dans les limites du périmètre, du temps, de la qualité et du budget impartis. La contribution du test au succès ne doit pas être limitée aux activités de l'équipe de test. Toute partie prenante peut utiliser ses compétences en matière de test pour rapprocher le projet de la réussite. Le test des composants, des systèmes et de la documentation associée permet d'identifier des défauts dans les logiciels.

1.2.1. Contributions du test au succès

Le test est un moyen efficace et peu coûteux de détecter des défauts. Ces défauts peuvent ensuite être éliminés (par débogage - une activité qui ne relève pas du test), de sorte que le test contribue indirectement à l'obtention d'objets de test de meilleure qualité.

Le test fournit un moyen d'évaluer directement la qualité d'un objet de test à différents stades du cycle de vie du développement logiciel. Ces mesures sont utilisées dans le cadre d'une activité de gestion de projet plus large, contribuant aux décisions de passer à l'étape suivante du cycle de vie du développement logiciel, telle que la décision de procéder à la livraison.

Le test offre aux utilisateurs une représentation indirecte du projet de développement. Les testeurs veillent à ce que leur compréhension des besoins des utilisateurs soit prise en compte tout au long du cycle de vie du développement logiciel. L'autre solution consiste à impliquer un ensemble représentatif d'utilisateurs dans le projet de développement, ce qui n'est généralement pas possible en raison des coûts élevés et du manque de disponibilité des utilisateurs adéquats.

Le test peut également être requis pour répondre à des exigences contractuelles ou légales, ou pour se conformer à des standards réglementaires.

1.2.2. Test et assurance qualité

Bien que les termes "test" et "assurance qualité" soient souvent utilisés de manière interchangeable, ils ne sont pas les mêmes. Le test est une forme de contrôle de la qualité.

Le contrôle de la qualité est une approche corrective axée sur le produit, qui se concentre sur les activités permettant d'atteindre des niveaux de qualité appropriés. Le test constitue une forme majeure de contrôle de la qualité, tandis que d'autres incluent les méthodes formelles (modèle de test et preuve d'exactitude), la simulation et le prototypage.

L'assurance qualité est une approche préventive axée sur les processus, qui se concentre sur la mise en œuvre et l'amélioration des processus. Elle part du principe que si un bon processus est suivi correctement, il générera un bon produit. L'assurance qualité s'applique à la fois aux processus de développement et de test, et relève de la responsabilité de tous les acteurs d'un projet.

Les résultats de test sont utilisés par l'assurance qualité et le contrôle de la qualité. Dans le cadre du contrôle de la qualité, ils servent à corriger les défauts, tandis que dans le cadre de l'assurance qualité, ils fournissent un retour d'information sur l'efficacité des processus de développement et de test.

1.2.3. Erreurs, défauts, défaillances et causes racine

Les êtres humains commettent des erreurs, qui produisent des défauts (fautes, bogues), lesquels peuvent à leur tour entraîner des défaillances. Les êtres humains commettent des erreurs pour diverses raisons, telles que la pression du temps, la complexité des produits d'activités, des processus, de l'infrastructure ou des interactions, ou simplement parce qu'ils sont fatigués ou qu'ils n'ont pas reçu une formation adéquate.

Les défauts peuvent se trouver dans la documentation, comme une spécification des exigences ou un script de test, dans le code source ou dans un artefact de support tel qu'un fichier de build. S'ils ne sont pas détectés, les défauts dans les artefacts produits au début du cycle de vie du développement logiciel conduisent souvent à des artefacts défectueux plus tard dans le cycle de vie. Si un défaut dans le code

est exécuté, le système peut ne pas faire ce qu'il devrait faire ou faire quelque chose qu'il ne devrait pas faire, ce qui entraîne une défaillance. Certains défauts entraîneront toujours une défaillance s'ils sont exécutés, tandis que d'autres n'entraîneront une défaillance que dans des circonstances spécifiques, et d'autres encore n'entraîneront jamais de défaillance.

Les erreurs et les défauts ne sont pas les seules causes de défaillance. Les défaillances peuvent également être causées par les conditions environnementales, par exemple lorsque les radiations ou les champs électromagnétiques provoquent des défauts dans les microprogrammes.

Une cause racine est une raison fondamentale de l'apparition d'un problème (par exemple, une situation qui conduit à une erreur). Les causes racines sont identifiées par l'analyse des causes racines, qui est généralement effectuée lorsqu'une défaillance se produit ou qu'un défaut est identifié. On estime qu'il est possible d'éviter d'autres défaillances ou défauts similaires ou de réduire leur fréquence en s'attaquant à la cause racine, par exemple en la supprimant.

1.3. Principes du test

Un certain nombre de principes du test, offrant des lignes directrices générales applicables à tous les tests, ont été proposés au fil des ans. Ce syllabus décrit sept de ces principes.

1. Le test montre la présence, et non l'absence, de défauts. Le test peut montrer que des défauts sont présents dans l'objet de test, mais ne peut pas prouver qu'il n'y a pas de défauts (Buxton 1970). Le test réduit la probabilité que des défauts ne soient pas découverts dans l'objet de test, mais même si aucun défaut n'est trouvé, le test ne peut pas prouver que l'objet de test est correct.

2. Le test exhaustif est impossible. Il n'est pas possible de tout tester, sauf dans les cas triviaux (Manna 1978). Plutôt que d'essayer de tester de manière exhaustive, les techniques de test (voir chapitre 4), la priorisation des cas de test (voir section 5.1.5) et le test basé sur les risques (voir section 5.2) doivent être utilisés pour concentrer les efforts de test.

3. Tester tôt économise du temps et de l'argent. Les défauts qui sont éliminés tôt dans le processus n'entraîneront pas de défauts ultérieurs dans les produits d'activités dérivés. Le coût de la qualité sera réduit puisque moins de défaillances se produiront plus tard dans le cycle du développement logiciel (Boehm 1981). Pour trouver les défauts le plus tôt possible, les tests statiques (voir chapitre 3) et les tests dynamiques (voir chapitre 4) doivent être lancés le plus tôt possible.

4. Regroupement des défauts. Un petit nombre de composants du système contiennent généralement la plupart des défauts découverts ou sont responsables de la plupart des défaillances opérationnelles (Enders 1975). Ce phénomène est une illustration du principe de Pareto. Les groupements de défauts prévus et les groupements de défauts réels observés pendant les tests ou en cours d'exploitation constituent une donnée d'entrée importante pour les tests basés sur les risques (voir section 5.2).

5. Usure des tests. Si les mêmes tests sont répétés plusieurs fois, ils deviennent de plus en plus inefficaces pour détecter de nouveaux défauts (Beizer 1990). Pour remédier à cet effet, il peut être nécessaire de modifier les tests et les données de test existants, et de rédiger de nouveaux tests. Toutefois, dans certains cas, la répétition des mêmes tests peut avoir un effet bénéfique, par exemple dans les tests de régression automatisés (voir section 2.2.3).

6. Le test dépend du contexte. Il n'existe pas d'approche de test unique et universellement applicable. Les tests sont effectués différemment selon les contextes (Kaner 2011).

7. L'illusion de l'absence de défaut. Il est illusoire (c'est-à-dire erroné) de penser que la vérification d'un logiciel garantira le succès d'un système. Si l'on teste minutieusement toutes les exigences spécifiées et que l'on corrige tous les défauts constatés, on peut néanmoins obtenir un système qui ne répond pas aux besoins et aux attentes des utilisateurs, qui ne permet pas d'atteindre les objectifs métier du client et qui est inférieur à d'autres systèmes concurrents. Outre la vérification, il convient également de procéder à la validation (Boehm 1981).

1.4. Activités de test, testware et rôles dans le test

Le test dépend du contexte, mais, à un niveau élevé, il existe des ensembles communs d'activités de test sans lesquels le test a moins de chances d'atteindre les objectifs de test. Ces ensembles d'activités de test forment un processus de test. Le processus de test peut être adapté à une situation donnée en fonction de divers facteurs. Les activités de test qui sont incluses dans ce processus de test, la manière dont elles sont implémentées et le moment où elles ont lieu sont normalement décidés dans le cadre de la planification des tests pour la situation spécifique (voir section 5.1).

Les sections suivantes décrivent les aspects généraux de ce processus de test en termes d'activités et de tâches de test, d'impact du contexte, de testware, de traçabilité entre la base de test et le testware, et de rôles de test.

Le standard ISO/IEC/IEEE 29119-2 fournit de plus amples informations sur les processus de test.

1.4.1. Activités et tâches de test

Un processus de test se compose généralement des principaux groupes d'activités décrits ci-dessous. Bien que nombre de ces activités semblent suivre une séquence logique, elles sont souvent mises en œuvre de manière itérative ou en parallèle. Ces activités de test doivent généralement être adaptées au système et au projet.

La planification des tests consiste à définir les objectifs du test, puis à sélectionner une approche qui permette d'atteindre au mieux les objectifs dans le cadre des contraintes imposées par le contexte général. La planification des tests est expliquée plus en détail au point 5.1.

Pilotage et contrôle des tests. Le pilotage des tests implique la vérification continue de toutes les activités de test et la comparaison des progrès réels par rapport au plan. Le contrôle des tests consiste à prendre les mesures nécessaires pour atteindre les objectifs du test. Le pilotage et le contrôle des tests sont expliqués plus en détail au point 5.3.

L'analyse de test comprend l'analyse de la base de test pour identifier les caractéristiques testables et pour définir et prioriser les conditions de test associées, ainsi que les risques et les niveaux de risque correspondants (voir section 5.2). La base de test et les objets de test sont également évalués pour identifier les défauts qu'ils peuvent contenir et pour évaluer leur testabilité. L'analyse de test est souvent soutenue par l'utilisation de techniques de test (voir chapitre 4). L'analyse de test répond à la question "que tester ?" en termes de critères de couverture mesurables.

La conception des tests comprend l'élaboration des conditions de test dans des cas de test et autres éléments du testware (par exemple, chartes de test). Cette activité implique souvent l'identification d'éléments de couverture, qui servent de guide pour spécifier les entrées des cas de test. Les techniques de test (voir chapitre 4) peuvent être utilisées pour soutenir cette activité. La conception des tests comprend également la définition des exigences en matière de données de test, la conception de

l'environnement de test et l'identification de toute autre infrastructure et de tout autre outil nécessaires. La conception des tests répond à la question "comment tester ?".

L'implémentation des tests comprend la création ou l'acquisition du testware nécessaire à l'exécution des tests (par exemple, les données de test). Les cas de test peuvent être organisés en procédures de test et sont souvent assemblés en suites de tests. Des scripts de test manuels et automatisés sont créés. Les procédures de test sont classées par ordre de priorité et organisées dans un calendrier d'exécution des tests pour une exécution efficiente des tests (voir section 5.1.5). L'environnement de test est construit et l'on vérifie qu'il est correctement configuré.

L'exécution des tests comprend l'exécution des tests conformément au calendrier d'exécution des tests. L'exécution des tests peut être manuelle ou automatisée. L'exécution des tests peut prendre de nombreuses formes, notamment celle du test en continu ou de sessions de test en binôme. Les résultats réels des tests sont comparés aux résultats attendus. Les résultats des tests sont enregistrés. Les anomalies sont analysées afin d'en identifier les causes probables. Cette analyse nous permet de signaler les anomalies sur la base des défaillances observées (voir section 5.5).

Les activités de clôture des tests ont généralement lieu lors des jalons du projet (par exemple, la livraison, la fin de l'itération, la clôture d'un niveau de test) pour tous les défauts non résolus, les demandes de changement ou les éléments créés dans le Product Backlog. Le testware qui pourrait être utile à l'avenir est identifié et archivé ou remis aux équipes appropriées. L'environnement de test est arrêté dans un état convenu. Les activités de test sont analysées afin d'identifier les leçons apprises et les améliorations à apporter aux futures itérations, versions ou projets (voir section 2.1.6). Un rapport d'achèvement des tests est créé et communiqué aux parties prenantes.

1.4.2. Le processus de test selon le contexte

Le test n'est pas effectué de manière isolée. Les activités de test font partie intégrante des processus de développement menés au sein d'une organisation. Le test est également financé par les parties prenantes et son objectif final est d'aider à répondre aux besoins métier des parties prenantes. Par conséquent, la manière dont le test est effectué dépendra d'un certain nombre de facteurs contextuels, notamment les suivants:

- Parties prenantes (besoins, attentes, exigences, volonté de coopérer, etc.)
- Membres de l'équipe (compétences, connaissances, niveau d'expérience, disponibilité, besoins de formation, etc.).
- Domaine d'activité (criticité de l'objet de test, risques identifiés, besoins métier, réglementations légales spécifiques, etc.).
- Facteurs techniques (type de logiciel, architecture du produit, technologie utilisée, etc.).
- Contraintes du projet (portée, temps, budget, ressources, etc.).
- Facteurs organisationnels (structure organisationnelle, politiques existantes, pratiques utilisées, etc.).
- Cycle de vie du développement logiciel (pratiques d'ingénierie, méthodes de développement, etc.).
- Outils (disponibilité, facilité d'utilisation, conformité, etc.). Ces facteurs auront un impact sur de nombreuses exigences liées aux tests, notamment : la stratégie de test, les techniques de test

utilisées, le degré d'automatisation des tests, le niveau de couverture requis, le niveau de détail de la documentation des tests, le reporting, etc.

-

1.4.3. Testware

Les composants du testware sont créés en tant que produits d'activités issus des activités de test décrites dans la section 1.4.1. La manière dont les différentes organisations produisent, façonnent, nomment, organisent et gèrent leurs produits d'activités varie considérablement. Une bonne gestion de la configuration (voir section 5.4) garantit la cohérence et l'intégrité des produits d'activités. La liste suivante de produits d'activités, classés par activité, n'est pas exhaustive:

- **Planification des tests**: plan de test, calendrier des tests, référentiel des risques et critères de sortie (voir section 5.1). Le référentiel des risques est une liste de risques accompagnée de la probabilité du risque, de l'impact du risque et d'informations sur l'atténuation des risques (voir section 5.2). Le calendrier des tests, le référentiel des risques et les critères d'entrée et de sortie font souvent partie du plan de test.
- **Pilotage et contrôle des tests**: rapports d'avancement des tests (voir point 5.3.2), documentation des directives de contrôle (voir point 5.3) et informations sur les risques (voir point 5.2).
- **Analyse de test**: conditions de test (priorisées) (par exemple, critères d'acceptation, voir section 4.5.2), et rapports de défauts concernant les défauts dans la base de test (s'ils ne sont pas corrigés directement).
- **Conception des tests**: cas de test (priorisés), chartes de test, éléments de couverture, exigences en matière de données de test et exigences en matière d'environnement de test.
- **Implémentation des tests**: procédures de test, scripts de test automatisés, suites de tests, données de test, calendrier d'exécution des tests, et éléments de l'environnement de test. Des éléments de l'environnement de test sont par exemple: les bouchons, les pilotes, les simulateurs et les virtualisations de services.
- **Exécution des tests**: logs de test et rapports de défaut (voir section 5.5).
- **Complétude des tests**: rapport de clôture des tests (voir section 5.3.2), mesures à prendre pour améliorer les projets ou itérations ultérieurs, leçons apprises documentées et demandes de changement (par exemple, en tant qu'éléments du product backlog).

1.4.4. Traçabilité entre base de test et testware

Afin d'implémenter un pilotage et un contrôle des tests efficaces, il est important d'établir et de maintenir la traçabilité tout au long du processus de test entre les éléments de la base de test, le testware associé à ces éléments (par exemple, les conditions de test, les risques, les cas de test), les résultats des tests et les défauts détectés

Une traçabilité précise soutient l'évaluation de la couverture, il est donc très utile que des critères de couverture mesurables soient définis dans la base de test. Les critères de couverture peuvent servir d'indicateurs de performance clés pour piloter les activités qui montrent dans quelle mesure les objectifs du test ont été atteints (voir section 1.1.1). Par exemple:

- La traçabilité des cas de test aux exigences permet de vérifier que les exigences sont couvertes par les cas de test.
- La traçabilité des résultats de test aux risques peut être utilisée pour évaluer le niveau de risque résiduel d'un objet de test.

Outre l'évaluation de la couverture, une bonne traçabilité permet de déterminer l'impact des changements, facilite les audits de test et aide à répondre aux critères de gouvernance informatique. Une bonne traçabilité rend également les rapports d'avancement et de clôture des tests plus facilement compréhensibles en incluant le statut des éléments de la base de test. Cela peut également aider à communiquer les aspects techniques des tests aux parties prenantes, de manière compréhensible. La traçabilité fournit des informations permettant d'auditer la qualité du produit, la capacité du processus et l'avancement du projet par rapport aux objectifs métiers.

1.4.5. Rôles dans le test

Dans ce syllabus, deux rôles principaux dans le test sont couverts : un rôle de test manager et un rôle de testeur. Les activités et les tâches assignées à ces deux rôles dépendent de facteurs tels que le contexte du projet et du produit, les compétences des personnes occupant ces rôles, et l'organisation.

Le rôle de test manager implique une responsabilité globale pour le processus de test, l'équipe de test et la direction des activités de test. Le rôle de test manager est principalement axé sur les activités de planification des tests, de pilotage et de contrôle des tests et de clôture des tests. La manière dont le rôle de test manager est exercé varie en fonction du contexte. Par exemple, dans le cadre du développement logiciel Agile, certaines des tâches du rôle de test manager peuvent être gérées par l'équipe Agile. Les tâches qui s'étendent à plusieurs équipes ou à l'ensemble de l'organisation peuvent être réalisées par des test managers en dehors de l'équipe de développement.

Le rôle de testeur implique une responsabilité globale pour l'aspect technique des tests. Le rôle de testeur est principalement axé sur les activités d'analyse de test, de conception des tests, d'implémentation des tests et d'exécution des tests.

Différentes personnes peuvent assumer ces rôles à différents moments. Par exemple, le rôle de test manager peut être assumé par un responsable de test, par un chef de projet de test, par un responsable des développements, etc. Il est également possible qu'une personne assume à la fois les rôles de testeur et de test manager.

1.5. Compétences essentielles et bonnes pratiques en matière de test

La compétence est l'aptitude à bien faire quelque chose qui découle des connaissances, de la pratique et des aptitudes d'une personne. Les bons testeurs doivent posséder certaines compétences essentielles pour bien faire leur travail. Les bons testeurs doivent être des membres efficaces d'une équipe et doivent être capables de mener des tests à différents niveaux d'indépendance du test.

1.5.1. Compétences génériques requise pour le test

Bien qu'elles soient génériques, les compétences suivantes sont particulièrement pertinentes pour les testeurs:

- Connaissance en matière de test (pour accroître l'efficacité des tests, par exemple en utilisant des techniques de test).
- Rigueur, attention, curiosité, souci du détail, méthode (pour identifier les défauts, en particulier ceux qui sont difficiles à trouver).
- Bonne communication, écoute active, esprit d'équipe (pour interagir efficacement avec toutes les parties prenantes, pour transmettre des informations aux autres, pour se faire comprendre, et pour signaler et discuter des défauts).
- Réflexion analytique, esprit critique, créativité (pour accroître l'efficacité des tests).
- Connaissances techniques (pour accroître l'efficience des tests, par exemple en utilisant les outils de test appropriés).
- Connaissance du domaine (pour être en mesure de comprendre les utilisateurs finaux/représentants métier et de communiquer avec eux).

Les testeurs sont souvent les porteurs de mauvaises nouvelles. C'est un trait humain courant que de blâmer le porteur de mauvaises nouvelles. C'est pourquoi les compétences en matière de communication sont cruciales pour les testeurs. La communication des résultats des tests peut être perçue comme une critique du produit et de son auteur. Le biais de confirmation peut rendre difficile l'acceptation d'informations en désaccord avec les convictions actuelles. Certaines personnes peuvent percevoir le test comme une activité destructrice, alors qu'il contribue grandement à la réussite du projet et à la qualité du produit. Pour tenter d'améliorer cette perception, les informations sur les défauts et les défaillances doivent être communiquées de manière constructive.

1.5.2. Approche équipe intégrée

L'une des compétences importantes pour un testeur est la capacité à travailler efficacement dans un contexte d'équipe et à contribuer positivement aux objectifs de l'équipe. L'approche équipe intégrée - une pratique issue de la programmation extrême (XP) (voir section 2.1) - repose sur cette compétence.

Dans l'approche intégrée, tout membre de l'équipe disposant des connaissances et des compétences nécessaires peut effectuer n'importe quelle tâche, et chacun est responsable de la qualité. Les membres de l'équipe partagent le même espace de travail (physique ou virtuel), car le regroupement facilite la communication et l'interaction. L'approche équipe intégrée améliore la dynamique d'équipe, la communication et la collaboration au sein de l'équipe, et crée une synergie en permettant aux différents ensembles de compétences au sein de l'équipe d'être exploités au profit du projet.

Les testeurs travaillent en étroite collaboration avec les autres membres de l'équipe afin de garantir que les niveaux de qualité souhaités sont atteints. Ils collaborent notamment avec les représentants métier pour les aider à créer des tests d'acceptation adaptés et travaillent avec les développeurs pour convenir de la stratégie de test et décider des approches d'automatisation des tests. Les testeurs peuvent ainsi transférer leurs connaissances en matière de test aux autres membres de l'équipe et influencer le développement du produit.

Selon le contexte, l'approche équipe intégrée n'est pas toujours appropriée. Par exemple, dans certaines situations, telles que les situations critiques pour la sécurité, un niveau élevé d'indépendance du test peut être nécessaire.

1.5.3. Indépendance du test

Un certain degré d'indépendance rend le testeur plus efficace pour trouver des défauts en raison des différences entre les biais cognitifs de l'auteur et ceux du testeur (cf. Salman 1995). L'indépendance ne remplace toutefois pas la proximité ; par exemple, les développeurs peuvent trouver efficacement de nombreux défauts dans leur propre code.

Les produits d'activités peuvent être testés par leur auteur (pas d'indépendance), par les pairs de l'auteur appartenant à la même équipe (un peu d'indépendance), par des testeurs extérieurs à l'équipe de l'auteur mais au sein de l'organisation (indépendance élevée), ou par des testeurs extérieurs à l'organisation (indépendance très élevée). Pour la plupart des projets, il est généralement préférable d'effectuer les tests avec plusieurs niveaux d'indépendance (par exemple, les développeurs effectuant les tests de composants et d'intégration de composants, l'équipe de test effectuant les tests de systèmes et d'intégration de systèmes, et les représentants métier effectuant les tests d'acceptation).

Le principal avantage de l'indépendance du test est que les testeurs indépendants sont susceptibles de repérer différents types de défaillances et de défauts par rapport aux développeurs en raison de leurs différents antécédents, perspectives techniques et biais. En outre, un testeur indépendant peut vérifier, contester ou réfuter les hypothèses formulées par les parties prenantes lors de la spécification et de la mise en œuvre du système.

Cependant, il y a aussi quelques inconvénients. Les testeurs indépendants peuvent être isolés de l'équipe de développement, ce qui peut entraîner un manque de collaboration, des problèmes de communication ou une relation d'opposition avec l'équipe de développement. Les développeurs peuvent perdre le sens de la responsabilité en matière de qualité. Les testeurs indépendants peuvent être considérés comme un goulot d'étranglement ou être tenus pour responsables des retards dans la release.

2. Tester tout au long du cycle de vie du développement logiciel – 130 minutes

Mots clés

tests d'acceptation, tests boîte noire, tests d'intégration de composants, tests de composants, tests de confirmation, tests fonctionnels, tests d'intégration, tests de maintenance, tests non fonctionnels, tests de régression, shift-left, tests d'intégration de systèmes, tests de systèmes, niveau de test, objet de test, type de test, tests boîte blanche.

Objectifs d'apprentissage pour le chapitre 2:

2.1 Tester dans le contexte d'un cycle de vie du développement logiciel

FL-2.1.1 (K2) Expliquer l'impact du cycle de vie du développement logiciel choisi sur le test

FL-2.1.2 (K1) Rappeler les bonnes pratiques de test qui s'appliquent à tous les cycles de vie du développement logiciel

FL-2.1.3 (K1) Rappeler des exemples d'approches de développement piloté par les tests

FL-2.1.4 (K2) Résumer la façon dont DevOps pourrait avoir un impact sur le test

FL-2.1.5 (K2) Expliquer l'approche shift-left

FL-2.1.6 (K2) Expliquer comment les rétrospectives peuvent être utilisées comme mécanisme d'amélioration des processus.

2.2 Niveaux de test et types de test

FL-2.2.1 (K2) Distinguer les différents niveaux de test

FL-2.2.2 (K2) Distinguer les différents types de tests

FL-2.2.3 (K2) Distinguer les tests de confirmation des tests de régression

2.3 Tests de maintenance

FL-2.3.1 (K2) Résumer les tests de maintenance et leurs déclencheurs

2.1. Tester dans le contexte d'un cycle de vie du développement logiciel

Un modèle de cycle de vie du développement logiciel est une représentation abstraite et de haut niveau du processus de développement logiciel. Un modèle de cycle de vie du développement logiciel définit la manière dont les différentes phases de développement et les types d'activités réalisées dans le cadre de ce processus sont liés les uns aux autres, à la fois logiquement et chronologiquement. Parmi les exemples de modèles, on peut citer : les modèles de développement séquentiel (par exemple, le modèle en cascade, le modèle en V), les modèles de développement itératif (par exemple, le modèle en spirale, le prototypage) et les modèles de développement incrémental (par exemple, le Processus Unifié).

Certaines activités au sein des processus de développement logiciel peuvent également être décrites par des méthodes de développement logiciel plus détaillées et des pratiques Agile. En voici quelques exemples : développement piloté par les tests d'acceptation (ATDD), développement piloté par le comportement (BDD), conception pilotée par le domaine (DDD), programmation extrême (XP), développement piloté par les fonctionnalités (FDD), Kanban, Lean IT, Scrum, et développement piloté par les tests (TDD).

2.1.1. Impact du cycle de vie du développement logiciel sur le test

Pour réussir, le test doit être adapté au cycle de vie du développement logiciel. Le choix du cycle de vie du développement logiciel a une incidence sur:

- Le périmètre et le calendrier des activités de test (par exemple, les niveaux de test et les types de test).
- Le niveau de détail de la documentation des tests.
- Le choix des techniques de test et de l'approche de test.
- Le degré d'automatisation des tests.
- Le rôle et les responsabilités d'un testeur.

Dans les modèles de développement séquentiel, au cours des phases initiales, les testeurs participent généralement à la revue des exigences, à l'analyse de test et à la conception des tests. Le code exécutable est généralement créé dans les phases ultérieures, de sorte que les tests dynamiques ne peuvent généralement pas être effectués au début du cycle de vie du développement logiciel.

Dans certains modèles de développement incrémental et itératif, on suppose que chaque itération fournit un prototype fonctionnant ou un incrément de produit. Cela implique qu'à chaque itération, des tests statiques et dynamiques peuvent être effectués à tous les niveaux de test. La livraison fréquente d'incréments nécessite un feedback rapide et des tests de régression poussés.

Le développement logiciel agile part du principe que des changements peuvent intervenir tout au long du projet. C'est pourquoi les projets agiles privilégient une documentation légère des produits d'activités et une automatisation poussée des tests pour faciliter les tests de régression. En outre, la plupart des tests manuels tendent à être effectués à l'aide de techniques de test basées sur l'expérience (voir section 4.4) qui ne nécessitent pas d'analyse et de conception des tests préalables approfondies.

2.1.2. Cycle de vie du développement logiciel et bonnes pratiques de test

Les bonnes pratiques de test, indépendantes du modèle de cycle de vie du développement logiciel choisi, préconisent notammentque:

- Pour chaque activité de développement logiciel, il existe une activité de test correspondante, de sorte que toutes les activités de développement sont soumises à un contrôle de qualité.
- Les différents niveaux de test (voir chapitre 2.2.1) ont des objectifs de test spécifiques et différents, ce qui permet aux tests d'être suffisamment complets tout en évitant la redondance.
- L'analyse et la conception des tests pour un niveau de test donné commencent pendant la phase de développement correspondante du cycle du développement logiciel, de sorte que les tests puissent adhérer au principe du test précoce (voir section 1.3).
- Les testeurs sont impliqués dans la revue des produits d'activités dès que les versions préliminaires de cette documentation sont disponibles, de sorte que ces tests et cette détection de défauts plus précoces puissent soutenir la stratégie shift left (voir section 2.1.5).

2.1.3. Le test en tant que moteur du développement de logiciels

TDD, ATDD et BDD sont des approches de développement similaires, dans lesquelles les tests sont définis comme un moyen d'orienter le développement. Chacune de ces approches implémente le principe des tests précoces (voir section 1.3) et suit une approche shift left (voir section 2.1.5), puisque les tests sont définis avant que le code ne soit écrit. Elles soutiennent un modèle de développement itératif. Ces approches se caractérisent comme suit:

Développement piloté par les tests (TDD):

- Dirige le codage par le biais de cas de tests (au lieu d'une conception approfondie du logiciel) (Beck 2003).
- Les tests sont écrits en premier, puis le code est écrit pour satisfaire les tests, et enfin les tests et le code sont refactorisés.

Développement piloté par les tests d'acceptation (ATDD) (voir section 4.5.3):

- Dérive des tests à partir des critères d'acceptation, dans le cadre du processus de conception des systèmes (Gärtner 2011).
- Les tests sont écrits avant que la partie de l'application ne soit développée pour satisfaire aux tests.

Développement piloté par le comportement (BDD) :

- Exprime le comportement souhaité d'une application avec des cas de test écrits dans une forme simple de langage naturel, qui est facile à comprendre par les parties prenantes - en utilisant généralement le format Given/When/Then (Etant donné/Lorsque/Alors). (Chelimsky 2010)
- Les cas de test sont ensuite automatiquement traduits en tests exécutables.

Pour toutes les approches ci-dessus, les tests peuvent persister en tant que tests automatisés afin de garantir la qualité du code lors des adaptations / refactorisations futures.

2.1.4. DevOps et tests

DevOps est une approche organisationnelle visant à créer une synergie en amenant le développement (y compris le test) et l'exploitation à travailler ensemble pour atteindre un ensemble d'objectifs communs. DevOps exige un changement culturel au sein d'une organisation pour combler les écarts entre le développement (y compris le test) et l'exploitation tout en traitant leurs fonctions avec la même valeur. DevOps favorise l'autonomie des équipes, le retour d'information rapide, les chaînes d'outils intégrées et les pratiques techniques telles que l'intégration continue (CI) et la livraison continue (CD). Cela permet aux équipes de construire, de tester et de livrer plus rapidement un code de haute qualité par le biais d'un pipeline de livraison DevOps (Kim 2016).

Du point de vue du test, voici quelques-uns des avantages de DevOps:

- Fournir un feedback rapide sur la qualité du code et sur l'impact éventuel des modifications sur le code existant.
- Favoriser une approche shift left du test (voir section 2.1.5) en encourageant les développeurs à soumettre un code de haute qualité accompagné de tests de composants et d'une analyse statique.
- Favoriser l'automatisation de processus tels que CI/CD qui facilitent la mise en place d'environnements de test stables.
- Augmenter la vue sur les caractéristiques qualité non fonctionnelles (par exemple, la performance, la fiabilité).
- Réduire le besoin de tests manuels répétitifs grâce à l'automatisation par le biais d'un pipeline de livraison.
- Minimiser le risque de régression grâce à l'ampleur et à la portée des tests de régression automatisés.

DevOps n'est pas sans risques et défis, lesquels incluent notamment que :

- Le pipeline de livraison DevOps doit être défini et établi.
- Les outils de CI/CD doivent être introduits et maintenus
- L'automatisation des tests nécessite des exigences supplémentaires et peut être difficile à mettre en place et à maintenir

Bien que DevOps s'accompagne d'un niveau élevé de tests automatisés, des tests manuels - en particulier du point de vue de l'utilisateur - seront toujours nécessaires.

2.1.5. Approche shift left

Le principe du test précoce (voir section 1.3) est parfois appelé shift left parce qu'il s'agit d'une approche dans laquelle le test est effectué plus tôt dans le cycle de vie du développement logiciel (SDLC). Le principe du shift left suggère normalement que les tests doivent être effectués plus tôt (par exemple, sans attendre que le code soit implémenté ou que les composants soient intégrés), mais il ne signifie pas que les tests doivent être négligés dans les phases ultérieures du cycle de vie du développement logiciel.

Il existe quelques bonnes pratiques qui illustrent comment réaliser un "shift left" dans le test :

- Examiner la spécification du point de vue des tests. Ces activités de revue des spécifications permettent souvent de trouver des défauts potentiels, tels que des ambiguïtés, des incomplétudes et des incohérences.
- Rédiger des cas de test avant l'écriture du code et faire exécuter le code dans un harnais de test pendant l'implémentation du code.
- Utiliser l'Intégration Continue et, mieux encore, le Développement Continu, permettant un feedback rapide et des tests de composants automatisés pour accompagner le code source lorsqu'il est déposé dans le référentiel de code.
- Clôturer les tests statiques du code source avant les tests dynamiques, ou dans le cadre d'un processus automatisé
- Effectuer des tests non fonctionnels en commençant par le niveau de test de composants, dans la mesure du possible. Il s'agit d'une forme de shift left, car ces types de tests non fonctionnels ont tendance à être réalisés plus tard dans le cycle de vie du développement logiciel, lorsqu'un système complet et un environnement de test représentatif sont disponibles.

Une approche shift left peut entraîner des formations, des efforts et/ou des coûts supplémentaires au début du processus, mais devrait permettre d'économiser des efforts et/ou des coûts à un stade ultérieur du processus.

Pour l'approche shift left, il est important que les parties prenantes soient convaincues et adhèrent à ce concept.

2.1.6. Rétrospectives et amélioration de processus

Les rétrospectives (également appelées "réunions post-projet" et rétrospectives de projet) sont souvent organisées à la fin d'un projet ou d'une itération, lors d'une étape de livraison, ou peuvent être organisées en cas de besoin. Le calendrier et l'organisation des rétrospectives dépendent du modèle de cycle de vie du développement logiciel suivi. Lors de ces réunions, les participants (non seulement les testeurs, mais aussi, par exemple, les développeurs, les architectes, le Product Owner, les analystes métier) discutent des points suivants:

- Qu'est-ce qui a été un succès et qui devrait être conservé ?
- Qu'est-ce qui n'a pas fonctionné et qui pourrait être amélioré ?
- Comment intégrer les améliorations et conserver les succès à l'avenir ?

Les résultats doivent être enregistrés et font normalement partie du rapport de clôture des tests (voir section 5.3.2). Les rétrospectives sont essentielles pour la mise en œuvre réussie de l'amélioration continue et il est important que toutes les améliorations recommandées soient suivies.

Les avantages typiques pour les tests sont les suivants:

- Augmentation de l'efficacité et de l'efficience des tests (par exemple, en mettant en œuvre des suggestions d'amélioration du processus de test).
- Augmentation de la qualité du testware (par exemple, en examinant conjointement les processus de test).

- Consolidation de l'équipe et apprentissage (par exemple, en raison de la possibilité de soulever des problèmes et de proposer des points d'amélioration).
- Amélioration de la qualité de la base de test (par exemple, car les lacunes dans l'étendue et la qualité des exigences peuvent être abordées et résolues).
- Amélioration de la coopération entre le développement et le test (par exemple, parce que la collaboration est régulièrement examinée et optimisée).

2.2. Niveaux de test et types de test

Les niveaux de test sont des groupes d'activités de test qui sont organisés et gérés ensemble. Chaque niveau de test est une instance du processus de test, exécutée en relation avec un logiciel à un stade de développement donné, depuis les 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 du développement logiciel. Dans les modèles séquentiels de cycle de vie du développement logiciel, les niveaux de test sont souvent définis de manière à ce que les critères de sortie d'un niveau fassent partie des critères d'entrée du niveau suivant. Dans certains modèles itératifs, cela peut ne pas s'appliquer. Les activités de développement peuvent s'étendre sur plusieurs niveaux de test. Les niveaux de test peuvent se chevaucher dans le temps.

Les types de test sont des groupes d'activités de test liées à des caractéristiques-qualité spécifiques et la plupart de ces activités de test peuvent être réalisées à chaque niveau de test.

2.2.1. Niveaux de test

Ce syllabus décrit les cinq niveaux de test suivants:

- **Les tests de composants** (également connus sous le nom de tests unitaires) se concentrent sur les tests de composants isolés. Ils nécessitent souvent un support spécifique, tel que des harnais de tests ou des frameworks de tests unitaires. Les tests de composants sont normalement effectués par les développeurs dans leur environnement de test.
- **Les tests d'intégration de composants** (également connus sous le nom de tests d'intégration unitaires) se concentrent sur le test des interfaces et des interactions entre les composants. Les tests d'intégration de composants dépendent fortement des approches de la stratégie d'intégration : ascendante, descendante ou big-bang.
- **Les tests système** se concentrent sur le comportement global et les capacités d'un système ou d'un produit entier, et comprennent souvent des tests fonctionnels, des tâches de bout en bout et des tests non fonctionnels de caractéristiques-qualité. Pour certaines caractéristiques-qualité non fonctionnelles, il est préférable de les tester sur un système complet dans un environnement de test représentatif (par exemple, utilisabilité). Il est également possible d'utiliser des simulateurs de sous-systèmes. Le test système peut être effectué par une équipe de test indépendante et est lié aux spécifications du système.
- **Les tests d'intégration du système** visent à tester les interfaces entre le système sous test et d'autres systèmes et services externes. Les tests d'intégration du système exigent des environnements de test appropriés, de préférence similaires à l'environnement opérationnel.

- **Les tests d'acceptation** sont axés sur la validation et la démonstration de l'aptitude au déploiement, ce qui signifie que le système répond aux besoins métier de l'utilisateur. Idéalement, les tests d'acceptation devraient être effectués par les utilisateurs prévus. Les principales formes de tests d'acceptation sont : les tests d'acceptation des utilisateurs (UAT), les tests d'acceptation opérationnelle, les tests d'acceptation contractuels et réglementaires, les tests alpha et les tests bêta**.**

Les niveaux de test se distinguent par la liste, non exhaustive, suivante d'attributs, afin d'éviter le chevauchement des activités de test:

- Objet de test.
- Objectifs du test.
- Base de test.
- Défauts et défaillances.
- Approche et responsabilités.

2.2.2. Types de test

Il existe de nombreux types de tests qui peuvent être appliqués dans les projets. Dans ce syllabus, les quatre types de tests suivants sont abordés:

Le test fonctionnel évalue les fonctions qu'un composant ou un système doit remplir. Les fonctions sont "ce que" l'objet de test doit faire. L'objectif principal du test fonctionnel est de vérifier la complétude fonctionnelle, l'exactitude fonctionnelle et l'adéquation fonctionnelle.

Le test non fonctionnel évalue les attributs autres que les caractéristiques fonctionnelles d'un composant ou d'un système. Le test non fonctionnel est le test de "comment le système se comporte". L'objectif principal du test non fonctionnel est de vérifier les caractéristiques-qualité non fonctionnelles du logiciel. Le standard ISO/IEC 25010 fournit la classification suivante des caractéristiques qualité non fonctionnelles des logiciels:

- Efficacité de la performance.
- Compatibilité.
- Utilisabilité.
- Fiabilité.
- Sécurité.
- Maintenabilité.
- Portabilité.

Il est parfois approprié que le test non fonctionnel commence tôt dans le cycle de vie du développement logiciel (par exemple, dans le cadre des revues et du test de composants ou du test de systèmes). De nombreux tests non fonctionnels sont dérivés des tests fonctionnels, car ils utilisent les mêmes tests fonctionnels, mais vérifient que lors de l'exécution de la fonction, une contrainte non fonctionnelle est satisfaite (par exemple, vérifier qu'une fonction s'exécute dans un délai spécifié, ou qu'une fonction peut être portée sur une nouvelle plateforme). La découverte tardive de défauts non fonctionnels peut

constituer une menace sérieuse pour la réussite d'un projet. Le test non fonctionnel nécessite parfois un environnement de test très spécifique, comme un laboratoire d'utilisabilité pour les tests d'utilisabilité.

Le test boîte noire (voir section 4.2) est basé sur les spécifications et dérive les tests de la documentation externe à l'objet de test. L'objectif principal du test boîte noire est de vérifier le comportement du système par rapport à ses spécifications.

Le test boîte blanche (voir section 4.3) est basé sur la structure et dérive les tests de l'implémentation ou de la structure interne du système (par exemple, le code, l'architecture, les flux métier et les flux de données). L'objectif principal du test boîte blanche est de couvrir la structure sous-jacente par les tests jusqu'au niveau acceptable.

Les quatre types de tests susmentionnés peuvent être appliqués à tous les niveaux de test, même si l'accent sera mis différemment à chaque niveau. Différentes techniques de test peuvent être utilisées pour dériver des conditions de test et des cas de test pour tous les types de test mentionnés.

2.2.3. Test de confirmation et test de régression

Des modifications sont généralement apportées à un composant ou à un système, soit pour l'améliorer par l'ajout d'une nouvelle caractéristique, soit pour le corriger par l'élimination d'un défaut. Les tests devraient alors également inclure des tests de confirmation et des tests de régression.

Le test de confirmation confirme qu'un défaut original a été corrigé avec succès. En fonction du risque, on peut tester la version corrigée du logiciel de plusieurs manières, notamment :

- En exécutant tous les cas de test qui ont précédemment échoué à cause du défaut, ou encore…
- En ajoutant de nouveaux tests pour couvrir les modifications nécessaires à la correction du défaut.

Toutefois, lorsque le temps ou l'argent manque pour corriger les défauts, le test de confirmation peut se limiter à l'exécution des étapes qui devraient reproduire la défaillance causée par le défaut et à la vérification que la défaillance ne se produit pas.

Le test de régression confirme qu'aucune conséquence négative n'a été causée par une modification, y compris un correctif qui a déjà fait l'objet d'un test de confirmation. Ces conséquences négatives pourraient affecter le composant où la modification a été apportée, d'autres composants du même système, voire d'autres systèmes connectés. Le test de régression peut ne pas se limiter à l'objet de test lui-même, mais peut également concerner l'environnement. Il est conseillé d'effectuer d'abord une analyse d'impact pour optimiser l'étendue du test de régression. L'analyse d'impact montre quelles parties du logiciel pourraient être affectées.

Les suites de test de régression sont exécutées de nombreuses fois et, en général, le nombre de cas de test de régression augmente à chaque itération ou release, de sorte que les tests de régression sont de bons candidats à l'automatisation. L'automatisation de ces tests devrait commencer dès le début du projet. Lorsque l'intégration continue est utilisée, comme dans DevOps (voir section 2.1.4), c'est une bonne pratique d'inclure également des tests de régression automatisés. Selon la situation, cela peut inclure des tests de régression à différents niveaux.

Des tests de confirmation et/ou de régression de l'objet du test sont nécessaires à tous les niveaux de test si des défauts sont corrigés et/ou si des modifications sont apportées à ces niveaux de test.

2.3. Test de maintenance

Il existe différentes catégories de maintenance : elle peut être corrective, s'adapter aux changements de l'environnement ou améliorer la performance ou la maintenabilité (voir ISO/IEC 14764 pour plus de détails). La maintenance peut donc impliquer des livraisons/déploiements planifiés et des livraisons/déploiements non planifiés (correctifs à chaud). Une analyse d'impact peut être effectuée avant une modification, pour aider à décider si le changement doit être effectué, sur la base des conséquences potentielles dans d'autres domaines du système. Le test des changements apportés à un système en production comprend à la fois l'évaluation du succès de l'implémentation des changements et la vérification des régressions possibles dans les parties du système qui restent inchangées (ce qui est généralement le cas de la majeure partie du système).

Le périmètre du test de maintenance dépend généralement :

- Du degré de risque des changements.
- De la taille du système de systèmes existant.
- De l'ampleur des modifications.

Les éléments déclencheurs de la maintenance et du test de maintenance peuvent être classés comme suit :

- Des modifications, telles que les améliorations planifiées (c'est-à-dire basées sur une livraison), les modifications correctives ou les correctifs à chaud.
- Des mises à niveau ou les migrations de l'environnement opérationnel, par exemple d'une plateforme à une autre, qui peuvent nécessiter des tests associés au nouvel environnement ainsi qu'au logiciel modifié, ou des tests de conversion des données lorsque les données d'une autre application sont migrées dans le système faisant l'objet de la maintenance.
- Le retrait, par exemple lorsqu'une application arrive en fin de vie. Le retrait d'un système peut nécessiter des tests d'archivage des données si de longues périodes de conservation des données sont requises. Le test des procédures de restauration et d'extraction après l'archivage peut également être nécessaire dans le cas où certaines données sont requises pendant la période d'archivage.

3. Test statique – 80 minutes

Mots clés

anomalie, test dynamique, revue formelle, revue informelle, inspection, revue, analyse statique, test statique, revue technique, relecture technique

Objectifs d'apprentissage pour le chapitre 3:

3.1 Bases du test statique

FL-3.1.1 (K1) Reconnaître les types de produits qui peuvent être examinés par les différentes techniques de test statique.

FL-3.1.2 (K2) Expliquer la valeur du test statique

FL-3.1.3 (K2) Comparer et opposer les tests statiques et les tests dynamiques.

3.2 Processus de feedback et de revue

FL-3.2.1 (K1) Identifier les avantages d'un feedback précoce et fréquent de la part des parties prenantes

FL-3.2.2 (K2) Résumer les activités du processus de revue

FL-3.2.3 (K1) Rappeler quelles sont les responsabilités attribuées aux rôles principaux lors des revues.

FL-3.2.4 (K2) Comparer et opposer les différents types de revues

FL-3.2.5 (K1) Rappeler les facteurs qui contribuent à la réussite d'une revue

3.1. Bases du test statique

Contrairement au test dynamique, le test statique ne nécessite pas l'exécution du logiciel testé. Le code, la spécification du processus, la spécification de l'architecture du système ou d'autres produits d'activités sont évalués par un examen manuel (par exemple, des revues) ou à l'aide d'un outil (par exemple, d'analyse statique). Les objectifs de test comprennent l'amélioration de la qualité, la détection des défauts et une évaluation des caractéristiques telles que la lisibilité, la complétude, la justesse, la testabilité et la cohérence. Le test statique peut être appliqué à la fois pour la vérification et la validation.

Les testeurs, les représentants métier et les développeurs travaillent ensemble pendant les sessions de cartographie d'exemples (Example mapping), de rédaction collaborative de User Stories et d' affinage du Backlog (Backlog refinement) pour s'assurer que les User Stories et les produits d'activités associés répondent aux critères définis, par exemple la Definition of Ready (voir section 5.1.3). Des techniques de revue peuvent être appliquées pour s'assurer que les User Stories sont complètes et compréhensibles et qu'elles incluent des critères d'acceptation testables. En posant les bonnes questions, les testeurs explorent, remettent en question et aident à améliorer les User Stories proposées.

L'analyse statique permet d'identifier les problèmes avant le test dynamique tout en demandant souvent moins d'efforts, puisqu'aucun cas de test n'est nécessaire et que des outils (voir chapitre 6) sont généralement utilisés. L'analyse statique est souvent incorporée dans des frameworks d'intégration continue (voir section 2.1.4). Bien qu'elle soit largement utilisée pour détecter des défauts de code spécifiques, l'analyse statique est également utilisée pour évaluer la maintenabilité et la sécurité. Les vérificateurs d'orthographe et les outils de lisibilité sont d'autres exemples d'outils d'analyse statique.

3.1.1. Produits d'activités examinables par le test statique

Presque tous les produits d'activités peuvent être examinés à l'aide du test statique. Les exemples incluent les documents de spécification des exigences, le code source, les plans de tests, les cas de tests, les éléments du product backlog, les chartes de tests, la documentation du projet, les contrats et les modèles.

Tout produit d'activités qui peut être lu et compris peut faire l'objet d'une revue. Toutefois, pour l'analyse statique, les produits d'activités ont besoin d'une structure par rapport à laquelle ils peuvent être vérifiés (par exemple, des modèles, du code ou du texte avec une syntaxe formelle).

Les produits d'activités qui ne conviennent pas au test statique sont ceux qui sont difficiles à interpréter par des êtres humains et qui ne devraient pas être analysés par des outils (par exemple, le code exécutable d'une tierce partie pour des raisons juridiques).

3.1.2. Valeur du test statique

Le test statique peut détecter des défauts dans les premières phases du cycle de vie du développement logiciel, répondant ainsi au principe du test précoce (voir section 1.3). Il permet également d'identifier les défauts qui ne peuvent pas être détectés par les tests dynamiques (par exemple, un code inaccessible, des canevas de conception qui ne sont pas implémentés comme souhaité, des défauts dans des produits d'activités non exécutables).

Le test statique permet d'évaluer la qualité des produits d'activités et de construire la confiance dans ces derniers. En vérifiant les exigences documentées, les parties prenantes peuvent également s'assurer que ces exigences décrivent leurs besoins réels. Comme le test statique peut être réalisé au début du cycle de vie du développement logiciel, il permet de créer une facilité de compréhension entre les parties

prenantes. La communication sera également améliorée entre les parties prenantes. C'est pourquoi il est recommandé d'impliquer une grande variété de parties prenantes dans le test statique.

Même si les revues peuvent être coûteuses à mettre en œuvre, les coûts globaux du projet seront généralement beaucoup moins élevés que lorsqu'aucune revue n'est effectuée, car moins de temps et d'efforts devront être consacrés à la correction de défauts à un stade ultérieur du projet.

Les défauts du code peuvent être détectés à l'aide de l'analyse statique de manière plus efficiente que dans le cadre d'un test dynamique, ce qui se traduit généralement par une réduction du nombre de défauts du code et de l'effort de développement global.

3.1.3. Différences entre le test statique et le test dynamique

Les pratiques de test statique et de test dynamique se complètent. Elles ont des objectifs similaires, tels que la détection des défauts dans les produits d'activités (voir section 1.1.1), mais il existe également des différences, telles que:

- Le test statique et le test dynamique (avec analyse des défaillances) peuvent tous deux conduire à la détection de défauts. Toutefois, certains types de défauts ne peuvent être trouvés que par le test statique ou le test dynamique.
- Le test statique constate directement les défauts, tandis que le test dynamique provoque des défaillances à partir desquelles les défauts associés sont déterminés par une analyse ultérieure.
- Le test statique permet de détecter plus facilement les défauts qui se trouvent sur des chemins du code rarement exécutés ou difficiles à atteindre par le test dynamique.
- Le test statique peut être appliqué à des produits d'activités non exécutables, alors que le test dynamique ne peut être appliqué qu'à des produits d'activités exécutables.
- Le test statique peut être utilisé pour mesurer les caractéristiques qualité (par exemple, la maintenabilité) qui ne dépendent pas de l'exécution du code, tandis que le test dynamique peut être utilisé pour mesurer les caractéristiques qualité (par exemple, l'efficacité de la performance) qui dépendent de l'exécution du code.

Les défauts typiques qu'il est plus facile et/ou moins coûteux de trouver par le biais du test statique comprennent:

- Des défauts dans les exigences (par exemple, incohérences, ambiguïtés, contradictions, omissions, inexactitudes, duplications).
- Des défauts de conception (par exemple, structures de base de données inefficaces, modularité insuffisante).
- Certains types de défauts de codage (par exemple, variables avec des valeurs non définies, variables non déclarées, code inaccessible ou dupliqué, complexité excessive du code).
- Des écarts par rapport aux standards (par exemple, le non-respect des conventions de nommage dans les standards de codage).

- Des spécifications d'interface incorrectes (par exemple, nombre, type ou ordre des paramètres non concordants).
- Des types spécifiques de vulnérabilités en matière de sécurité (par exemple, des débordements de mémoire tampon).
- Des lacunes ou des imprécisions dans la couverture de la base de test (par exemple, des tests manquants pour un critère d'acceptation).

3.2. Processus de feedback et de revue

3.2.1. Bénéfices d'un feedback précoce et fréquent des parties prenantes

Un feedback précoce et fréquent permet de communiquer rapidement les problèmes de qualité potentiels. Si les parties prenantes sont peu impliquées dans le cycle de vie du développement logiciel, le produit développé risque de ne pas correspondre à la vision initiale ou actuelle des parties prenantes. Une défaillance dans la réalisation des souhaits de la partie prenante peut entraîner des rectifications coûteuses, des délais non respectés, des (jeux de) reproches, voire l'échec total du projet.

Un feedback fréquent des parties prenantes tout au long du cycle de vie du développement logiciel permet d'éviter les malentendus sur les exigences et de s'assurer que les changements apportés aux exigences sont compris et mis en œuvre plus tôt. Cela permet à l'équipe de développement d'améliorer sa compréhension de ce qu'elle est en train de construire. Elle peut ainsi se concentrer sur les caractéristiques qui apportent le plus de valeur aux parties prenantes et qui ont l'impact le plus positif sur les risques identifiés.

3.2.2. Activités du processus de revue

Le standard ISO/IEC 20246 définit un processus de revue générique qui fournit un cadre structuré mais flexible à partir duquel un processus de revue spécifique peut être adapté à une situation particulière. Si la revue requise est plus formelle, un plus grand nombre de tâches pour les différentes activités seront nécessaires.

La taille de nombreux produits d'activités les rend trop volumineux pour être couverts par un seul réviseur. Le processus de revue peut être invoqué à plusieurs reprises pour achever la revue de l'ensemble du produit d'activités.

Les activités du processus de revue sont les suivantes:

- **Planification.** Au cours de la phase de planification, le périmètre de la revue, qui comprend l'objectif, le produit d'activités à examiner, les caractéristiques-qualité à évaluer, les domaines à privilégier, les critères de sortie, les informations complémentaires (telles que les standards), l'effort et les délais de la revue ; doit être défini.
- **Lancement de la revue.** Lors du lancement de la revue, l'objectif est de s'assurer que toutes les personnes impliquées sont prêtes à commencer la revue. Il s'agit notamment de s'assurer que chaque participant a accès au produit d'activités examiné, qu'il comprend son rôle et ses responsabilités et qu'il reçoit tout ce qui est nécessaire à l'exécution de la revue.

- **Revue individuelle.** Chaque réviseur effectue une revue individuelle afin d'évaluer la qualité du produit d'activités en cours de revue et d'identifier les anomalies, les recommandations et des questions en appliquant une ou plusieurs techniques de revue (par exemple, revue basée sur des checklists, revue basée sur des scénarios). La norme ISO/IEC 20246 fournit des informations plus détaillées sur les différents types de revues. Les réviseurs enregistrent toutes les anomalies, recommandations et questions qu'ils ont identifiées.
- **Communication et analyse.** Étant donné que les anomalies identifiées lors d'une revue ne sont pas nécessairement des défauts, toutes ces anomalies doivent être analysées et discutées. Pour chaque anomalie, il convient de prendre une décision sur son statut, son responsable et les actions requises. Cela se fait généralement lors d'une réunion de revue, au cours de laquelle les participants décident également du niveau de qualité du produit d'activités revu et des actions de suivi nécessaires. Une revue de suivi peut être nécessaire pour mener à bien les actions.
- **Correction et rapport.** Pour chaque défaut, un rapport de défaut doit être créé afin que les actions correctives puissent faire l'objet d'un suivi. Lorsque les critères de sortie sont atteints, le produit d'activités peut être accepté. Les résultats de la revue font l'objet d'un rapport.

3.2.3. Rôles et responsabilités dans les revues

Les revues impliquent diverses parties prenantes, qui peuvent jouer plusieurs rôles. Les principaux rôles et leurs responsabilités sont les suivants:

- **Manager -** décide de ce qui doit être revu et fournit les ressources, telles que le personnel et le temps nécessaires à la réalisation de la revue.
- **Auteur -** crée et corrige le produit d'activités en cours de revue.
- **Modérateur (également appelé facilitateur)** - veille au déroulement efficace des réunions de revue, y compris la médiation, la gestion du temps et un environnement de revue sûr dans lequel chacun peut s'exprimer librement.
- **Scribe (également appelé rapporteur)** - rassemble les anomalies provenant des réviseurs et enregistre les informations relatives à la revue, telles que les décisions et les nouvelles anomalies trouvées au cours de la réunion de revue.
- **Réviseur** - effectue les revues. Un réviseur peut être une personne travaillant sur le projet, un expert en la matière ou toute autre partie prenante.
- **Responsable de la revue** - assume la responsabilité générale de la revue, notamment en décidant qui sera impliqué et en organisant le lieu et la date de la revue.

D'autres rôles plus détaillés sont possibles, comme le décrit le standard ISO/IEC 20246.

3.2.4. Types de revues

Il existe de nombreux types de revues, allant de la revue informelle à la revue formelle. Le niveau de formalité requis dépend de facteurs tels que le cycle de vie du développement logiciel suivi, la maturité du processus de développement, la criticité et la complexité du produit d'activités examiné, les exigences légales ou réglementaires et les exigences liées à un possible audit. Le même produit d'activités peut être revu avec différents types de revues, par exemple d'abord une revue informelle, puis une revue plus formelle.

Le choix du bon type de revue est essentiel pour atteindre les objectifs de revue requis (voir section 3.2.5). La sélection n'est pas seulement pilotée par les objectifs, mais aussi par des facteurs tels que les besoins du projet, les disponibilités, le type de produit d'activités et les risques, le domaine d'activité et la culture de l'entreprise.

Voici des types de revues couramment utilisés:

- **Revue informelle.** Les revues informelles ne suivent pas un processus défini et n'exigent pas de résultat formel documenté. L'objectif principal est de détecter des anomalies.
- **Relecture technique.** Une relecture technique, menée par l'auteur, peut répondre à de nombreux objectifs, tels que l'évaluation de la qualité et le renforcement de la confiance dans le produit d'activités, la formation des réviseurs, l'obtention d'un consensus, la génération de nouvelles idées, la motivation et la capacité des auteurs à s'améliorer et la détection d'anomalies. Les réviseurs peuvent effectuer une revue individuelle avant la relecture technique, mais ce n'est pas obligatoire.
- **Revue technique.** Une revue technique est réalisée par des réviseurs techniquement qualifiés et dirigée par un modérateur. Les objectifs d'une revue technique sont de parvenir à un consensus et de prendre des décisions concernant un problème technique, mais aussi de détecter des anomalies, d'évaluer la qualité et de construire la confiance dans le produit d'activités, de générer de nouvelles idées, de motiver les auteurs et de leur permettre de s'améliorer.
- **L'inspection.** Les inspections étant le type de revue le plus formel, elles suivent le processus générique complet (voir section 3.2.2). L'objectif principal est de trouver le maximum d'anomalies. D'autres objectifs sont d'évaluer la qualité, de construire la confiance dans le produit d'activités, de motiver les auteurs et de leur permettre de s'améliorer. Des métriques sont collectées et utilisées pour améliorer le cycle de vie de développement du logiciel, y compris le processus d'inspection. Lors des inspections, l'auteur ne peut pas jouer le rôle de réviseur ou de scribe.

3.2.5. Facteurs de réussite des revues

Plusieurs facteurs déterminent le succès des revues, notamment :

- Définir des objectifs clairs et des critères de sortie mesurables. L'évaluation des participants ne doit jamais être un objectif.
- Choisir le type de revue approprié pour atteindre les objectifs fixés et s'adapter au type de produit d'activités, aux participants à la revue, aux besoins du projet et au contexte.
- Réaliser des revues en petits groupes, afin que les réviseurs ne se déconcentrent pas au cours d'une revue individuelle et/ou de la réunion de revue (le cas échéant).
- Fournir un feedback des revues aux parties prenantes et aux auteurs afin qu'ils puissent améliorer le produit et leurs activités (voir section 3.2.1).
- Accorder aux participants suffisamment de temps pour se préparer à la revue.
- Bénéficier d'un soutien, de la part du management, au processus de revue.
- Intégrer les revues dans la culture de l'organisation, afin de promouvoir l'apprentissage et l'amélioration des processus.

- Fournir une formation adéquate à tous les participants afin qu'ils sachent comment remplir leur rôle.
- Faciliter les réunions.

4. Analyse et conception des tests – 390 minutes

Mots clés

critères d'acceptation, développement piloté par les tests d'acceptation, technique de test boîte noire, analyse des valeurs limites, couverture des branches, test basé sur une checklist, approche de test basée sur la collaboration, couverture, élément de couverture, test basé sur une table de décision, partition d'équivalence, estimation d'erreurs, technique de test basée sur l'expérience, test exploratoire, test des transitions d'état, couverture des instructions, technique de test, technique de test boîte blanche

Objectifs d'apprentissage pour le chapitre 4:

4.1 Aperçu des techniques de test

FL-4.1.1 (K2) Distinguer 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) Utiliser les partitions d'équivalence pour dériver les cas de test

FL-4.2.2 (K3) Utiliser l'analyse des valeurs limites pour dériver les cas de test

FL-4.2.3 (K3) Utiliser les tests par tables de décisions pour dériver les cas de test.

FL-4.2.4 (K3) Utiliser les tests de transition d'état pour dériver les cas de test.

4.3 Techniques de test boîte-blanche

FL-4.3.1 (K2) Expliquer le test des instructions

FL-4.3.2 (K2) Expliquer le test des branches

FL-4.3.3 (K2) Expliquer la valeur des tests boîte blanche

4.4 Techniques de test basées sur l'expérience

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

FL-4.4.2 (K2) Expliquer le test exploratoire

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

4.5. Approches de test basées sur la collaboration

FL-4.5.1 (K2) Expliquer comment rédiger des User Stories en collaboration avec des développeurs et des représentants du métier

FL-4.5.2 (K2) Classer les différentes options pour la rédaction des critères d'acceptation

FL-4.5.3 (K3) Utiliser le développement piloté par les tests d'acceptation (ATDD) pour dériver les cas de test.

4.1. Aperçu des techniques de test

Les techniques de test aident le testeur dans l'analyse de test (ce qu'il faut tester) et dans la conception des tests (comment tester). Les techniques de test aident à développer de manière systématique un ensemble relativement restreint, mais suffisant, de cas de test. Les techniques de test aident également le testeur à définir les conditions de test, à identifier les éléments de couverture et à identifier les données de test au cours de l'analyse et de la conception des tests. De plus amples informations sur les techniques de test et les mesures correspondantes peuvent être trouvées dans le standard ISO/IEC/IEEE 29119-4, et dans (Beizer 1990, Craig 2002, Copeland 2004, Koomen 2006, Jorgensen 2014, Ammann 2016, Forgács 2019).

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

Les techniques de test boîte noire (également connues sous le nom de techniques basées sur les spécifications) sont basées sur une analyse du comportement spécifié de l'objet de test sans référence à sa structure interne. Les cas de test sont donc indépendants de l'implémentation du logiciel. Par conséquent, si l'implémentation change, mais que le comportement requis reste le même, les cas de test restent utiles.

Les techniques de test boîte blanche (également connues sous le nom de techniques basées sur la structure) sont basées sur une analyse de la structure interne et du traitement de l'objet de test. Comme les cas de test dépendent de la conception du logiciel, ils ne peuvent être créés qu'après la conception ou l'implémentation de l'objet de test.

Les techniques de test basées sur l'expérience utilisent efficacement les connaissances et l'expérience des testeurs pour la conception et l'implémentation des cas de test. L'efficacité de ces techniques dépend fortement des compétences du testeur. Les techniques de test basées sur l'expérience peuvent détecter des défauts qui pourraient échapper aux techniques de test boîte noire et boîte blanche. Les techniques de test basés sur l'expérience sont donc complémentaires des techniques de test boîte noire et boîte blanche.

4.2. Techniques de test boîte noire

Les techniques de test boîte noire couramment utilisées et présentées dans les sections suivantes sont les suivantes :

- Partitions d'équivalence.
- Analyse des valeurs limites.
- Test par tables de décisions.
- Test de transition d'état.

4.2.1. Partitions d'équivalence

Les partitions d'équivalence divisent les données en partitions (appelées partitions d'équivalence) en partant du principe que tous les éléments d'une partition donnée doivent être traités de la même manière par l'objet de test. La théorie qui sous-tend cette technique est que si un cas de test, qui teste une valeur

d'une partition d'équivalence, détecte un défaut, ce défaut devrait également être détecté par les cas de test qui testent n'importe quelle autre valeur de la même partition. Par conséquent, un test pour chaque partition est suffisant.

Les partitions d'équivalence peuvent être identifiées pour tout élément de données lié à l'objet de test, y compris les entrées, les sorties, les éléments de configuration, les valeurs internes, les valeurs liées au temps et les paramètres d'interface. Les partitions peuvent être continues ou discrètes, ordonnées ou non, finies ou infinies. Les partitions ne doivent pas se chevaucher et doivent être des ensembles non vides.

Pour les objets de test simples, la technique des partitions d'équivalence peut être simple, mais dans la pratique, il est souvent compliqué de comprendre comment l'objet de test traitera les différentes valeurs. C'est pourquoi le partitionnement doit être effectué avec soin.

Une partition contenant des valeurs valides est appelée partition valide. Une partition contenant des valeurs invalides est appelée partition invalide. Les définitions des valeurs valides et invalides peuvent varier selon les équipes et les organisations. Par exemple, les valeurs valides peuvent être interprétées comme celles qui doivent être traitées par l'objet de test ou comme celles pour lesquelles la spécification définit le traitement. Les valeurs invalides peuvent être interprétées comme celles qui doivent être ignorées ou rejetées par l'objet de test ou comme celles pour lesquelles aucun traitement n'est défini dans la spécification de l'objet de test.

Dans la technique des partitions d'équivalence, les éléments de couverture sont les partitions d'équivalence. Pour atteindre une couverture de 100 % avec cette technique, les cas de test doivent exercer toutes les partitions identifiées (y compris les partitions invalides) en couvrant chaque partition au moins une fois. La couverture est mesurée comme le nombre de partitions exercées par au moins un cas de test, divisé par le nombre total de partitions identifiées, et est exprimée en pourcentage.

De nombreux objets de test comprennent plusieurs ensembles de partitions (par exemple, les objets de test avec plus d'un paramètre d'entrée), ce qui signifie qu'un cas de test couvrira des partitions provenant de différents ensembles de partitions. Le critère de couverture le plus simple dans le cas d'ensembles multiples de partitions est appelé couverture Each Choice (Ammann 2016). La couverture Each Choice exige que les cas de test exercent chaque partition de chaque ensemble de partitions au moins une fois. La couverture Each Choice ne prend pas en compte les combinaisons de partitions.

4.2.2. Analyse des valeurs limites

L'analyse des valeurs limites est une technique basée sur l'exercice des limites des partitions d'équivalence. Par conséquent, la technique de l'analyse des valeurs limites ne peut être utilisée que pour des partitions ordonnées. Les valeurs minimales et maximales d'une partition sont ses valeurs limites. Dans le cas de la technique de l'analyse des valeurs limites, si deux éléments appartiennent à la même partition, tous les éléments qui se trouvent entre eux doivent également appartenir à cette partition.

La technique de l'analyse des valeurs limites se concentre sur les valeurs limites des partitions, car les développeurs sont plus susceptibles de commettre des erreurs au niveau de ces valeurs limites. Les défauts typiques trouvés par cette technique se situent là où les limites implémentées sont mal placées, à des positions supérieures ou inférieures à celles prévues, ou sont complètement omises.

Ce syllabus couvre deux versions de la technique de l'analyse des valeurs limites : la technique à 2 valeurs et la technique à 3 valeurs. Elles diffèrent en termes d'éléments de couverture par limite qui doivent être exercés pour atteindre une couverture de 100 %.

Dans la technique à 2 valeurs (Craig 2002, Myers 2011), pour chaque valeur limite, il y a deux éléments de couverture : la valeur limite en question et sa voisine la plus proche appartenant à la partition adjacente. Pour atteindre une couverture de 100 % avec la technique à 2 valeurs, les cas de test doivent exercer tous les éléments de couverture, c'est-à-dire toutes les valeurs limites identifiées. La couverture est mesurée comme le nombre de valeurs limites exercées, divisé par le nombre total de valeurs limites identifiées, et est exprimée en pourcentage.

Dans la technique à 3 valeurs (Koomen 2006, O'Regan 2019), pour chaque valeur limite, il y a trois éléments de couverture : la valeur limite en question et ses deux voisines. Par conséquent, dans la technique à 3 valeurs, certains des éléments de couverture peuvent ne pas être des valeurs limites. Pour atteindre une couverture de 100 % avec la technique à 3 valeurs, les cas de test doivent exercer tous les éléments de couverture, c'est-à-dire les valeurs limites identifiées et leurs voisines. La couverture est mesurée comme le nombre de valeurs limites et de leurs voisines exercées, divisé par le nombre total de valeurs limites identifiées et de leurs voisines, et est exprimée en pourcentage.

La technique à 3 valeurs est plus rigoureuse que la technique à 2 valeurs, car elle permet de détecter des défauts qui n'ont pas été pris en compte par la technique à 2 valeurs. Par exemple, si la décision "si ($x \leq 10$) ..." est incorrectement implémentée comme "si ($x = 10$) ...", aucune donnée de test dérivée de la technique à 2 valeurs ($x = 10$, $x = 11$) ne peut détecter le défaut. Cependant, $x = 9$, dérivé de la technique à 3 valeurs, est susceptible de le détecter.

4.2.3. Test par tables de décisions

Les tables de décision sont utilisées pour tester l'implémentation des exigences du système qui spécifient comment différentes combinaisons de conditions aboutissent à des résultats différents. Les tables de décision constituent un moyen efficace d'enregistrer une logique complexe, telle que les règles de gestion.

Lors de la création de tables de décision, les conditions et les actions résultantes du système sont définies. Elles constituent les lignes de la table. Chaque colonne correspond à une règle de décision qui définit une combinaison unique de conditions, ainsi que les actions associées. Dans les tables de décision à entrée limitée, toutes les valeurs des conditions et des actions (à l'exception des valeurs non pertinentes ou irréalisables ; voir ci-dessous) sont présentées sous forme de valeurs booléennes (vrai ou faux). Dans les tables de décision à entrée étendue, tout ou partie des conditions et des actions peuvent également prendre des valeurs multiples (plages de nombres, partitions d'équivalence, valeurs discrètes, etc.)

La notation des conditions est la suivante : " V " (vrai) signifie que la condition est remplie. "F" (faux) signifie que la condition n'est pas remplie. "-" signifie que la valeur de la condition n'est pas pertinente pour le résultat de l'action. "N/A" signifie que la condition est irréalisable pour une règle donnée. Pour les actions : "X" signifie que l'action doit avoir lieu. Blanc signifie que l'action ne doit pas se produire. D'autres notations peuvent également être utilisées.

Une table de décision complète comporte suffisamment de colonnes pour couvrir toutes les combinaisons de conditions. La table peut être simplifiée en supprimant les colonnes contenant des combinaisons de conditions irréalisables. La table peut également être minimisée en fusionnant les colonnes dans lesquelles certaines conditions n'affectent pas le résultat en une seule colonne. Les algorithmes de minimisation des tables de décision sont hors du périmètre de ce syllabus.

Dans les tests par tables de décisions, les éléments de couverture sont les colonnes contenant les combinaisons de conditions réalisables. Pour atteindre une couverture de 100 % avec cette technique, les cas de test doivent exercer toutes ces colonnes.

La couverture est mesurée comme le nombre de colonnes exercées, divisé par le nombre total de colonnes réalisables, et est exprimée en pourcentage.

La force du test par tables de décisions réside dans le fait qu'il fournit une approche systématique permettant d'identifier toutes les combinaisons de conditions, dont certaines pourraient autrement être négligées. Il permet également de trouver des lacunes ou des contradictions dans les exigences. Si les conditions sont nombreuses, l'exercice de toutes les règles de décision peut prendre beaucoup de temps, car le nombre de règles croît de manière exponentielle avec le nombre de conditions. Dans ce cas, pour réduire le nombre de règles à appliquer, on peut utiliser une table de décision réduite ou une approche basée sur les risques.

4.2.4. Test de transition d'état

Un diagramme de transition d'états modélise le comportement d'un système en montrant ses états possibles et les transitions d'états valides. Une transition est initiée par un événement, qui peut être qualifié par une condition de garde. Les transitions sont supposées être instantanées et peuvent parfois entraîner une action du logiciel. La syntaxe courante utilisée pour désigner les transitions est la suivante : "événement [condition de garde] / action". Les conditions de garde et les actions peuvent être omises si elles n'existent pas ou si elles ne sont pas pertinentes pour le testeur.

Une table d'états est un modèle équivalent à un diagramme de transition d'états. Ses lignes représentent les états et ses colonnes les événements (ainsi que les conditions de garde si elles existent). Les entrées de la table (cellules) représentent les transitions et contiennent l'état cible, ainsi que les conditions de garde et les actions résultantes, si elles sont définies. Contrairement au diagramme de transition d'états, la table d'états montre explicitement les transitions non valides, qui sont représentées par des cellules vides.

Un cas de test basé sur un diagramme de transition d'état ou une table d'états est généralement représenté comme une séquence d'événements, qui résulte en une séquence de changements d'état (et d'actions, si nécessaire). Un cas de test peut, et c'est généralement le cas, couvrir plusieurs transitions d'états.

Il existe de nombreux critères de couverture pour les tests de transition d'état. Ce syllabus discute de trois d'entre eux.

Dans la **couverture de tous les états**, les éléments de couverture sont les états. Pour obtenir une couverture de 100 %, les cas de test doivent garantir que tous les états sont visités. La couverture est mesurée comme le nombre d'états visités divisé par le nombre total d'états, et est exprimée en pourcentage.

Dans la **couverture des transitions valides** (également appelée couverture 0-switch), les éléments de couverture sont les transitions valides uniques. Pour obtenir une couverture de 100% des transitions valides, les cas de test doivent exercer toutes les transitions valides. La couverture est mesurée comme le nombre de transitions valides exercées divisé par le nombre total de transitions valides, et est exprimée en pourcentage.

Dans la **couverture de toutes les transitions**, les éléments de couverture sont toutes les transitions figurant dans une table d'état. Pour obtenir une couverture de 100 % de toutes les transitions, les cas de

test doivent exercer toutes les transitions valides et tenter d'exécuter des transitions non valides. Le fait de ne tester qu'une seule transition non valide dans un seul cas de test permet d'éviter le masquage des défauts, c'est-à-dire une situation dans laquelle un défaut empêche la détection d'un autre. La couverture est mesurée comme le nombre de transitions valides et non valides exercées ou tentées d'être exercées par les cas de test exécutés, divisé par le nombre total de transitions valides et non valides, et est exprimée en pourcentage.

La couverture de tous les états est plus faible que la couverture des transitions valides, car elle peut généralement être obtenue sans exercer toutes les transitions. La couverture des transitions valides est le critère de couverture le plus largement utilisé. L'obtention d'une couverture complète des transitions valides garantit une couverture complète de tous les états. L'obtention d'une couverture complète de toutes les transitions garantit à la fois une couverture complète de tous les états et une couverture complète des transitions valides et devrait constituer une exigence minimale pour les logiciels critiques en termes de mission et de sécurité.

4.3. Techniques de test boîte blanche

En raison de leur popularité et de leur simplicité, cette section se concentre sur deux techniques de test boîte blanche liées au code :

- Le test des instructions.
- Le test des branches.

Il existe des techniques plus rigoureuses qui sont utilisées dans certains environnements critiques en termes de sécurité, de mission ou d'intégrité élevée afin d'obtenir une couverture plus complète du code. Il existe également des techniques de tests boîte blanche utilisées à des niveaux de test plus élevés (par exemple, tests API), ou utilisant une couverture non liée au code (par exemple, la couverture des neurones dans les tests de réseaux neuronaux). Ces techniques ne sont pas abordées dans ce syllabus.

4.3.1. Test des instructions et couverture des instructions

Dans le test des instructions, les éléments de couverture sont des instructions exécutables. L'objectif est de concevoir des cas de test qui exercent les instructions du code jusqu'à ce qu'un niveau de couverture acceptable soit atteint. La couverture est mesurée comme le nombre d'instructions exercées par les cas de test divisé par le nombre total d'instructions exécutables dans le code, et est exprimée en pourcentage.

Lorsque la couverture des instructions est de 100 %, cela signifie que toutes les instructions exécutables du code ont été testées au moins une fois. En particulier, cela signifie que chaque instruction comportant un défaut sera exécutée, ce qui peut provoquer une défaillance démontrant la présence du défaut. Cependant, exécuter une instruction avec un cas de test ne permet pas de détecter les défauts dans tous les cas. Par exemple, il se peut qu'il ne détecte pas les défauts qui dépendent des données (par exemple, une division par zéro qui n'échoue que lorsque le dénominateur est fixé à zéro). De même, une couverture des instructions à 100 % ne garantit pas que toute la logique de décision a été testée, car, par exemple, toutes les branches (voir le chapitre 4.3.2) du code n'ont pas été testées.

4.3.2. Test des branches et couverture des branches

Une branche est un transfert de contrôle entre deux nœuds dans le graphe de flux de contrôle, qui montre les séquences possibles dans lesquelles les instructions du code source sont exécutées dans l'objet de test. Chaque transfert de contrôle peut être inconditionnel (c'est-à-dire un code en ligne droite) ou conditionnel (c'est-à-dire un résultat de décision).

Dans les tests de branches, les éléments de couverture sont les branches et l'objectif est de concevoir des tests pour exercer les branches du code jusqu'à ce qu'un niveau de couverture acceptable soit atteint. La couverture est mesurée comme le nombre de branches exercées par les cas de test divisé par le nombre total de branches, et est exprimée en pourcentage.

Lorsque la couverture des branches est de 100 %, toutes les branches du code, qu'elles soient inconditionnelles ou conditionnelles, sont exercées par les cas de test. Les branches conditionnelles correspondent généralement à un résultat vrai ou faux d'une décision "if...then", à un résultat d'une instruction "switch/case", ou à une décision de quitter ou de continuer dans une boucle. Cependant, l'exercice d'une branche avec un cas de test ne permet pas de détecter les défauts dans tous les cas. Par exemple, il peut ne pas détecter les défauts nécessitant l'exécution d'un chemin spécifique dans le code.

La couverture des branches englobe la couverture des instructions. Cela signifie que tout ensemble de cas de test atteignant une couverture des branches de 100 % atteint également une couverture des instructions de 100 % (mais pas l'inverse).

4.3.3. La valeur des tests boîte blanche

Une force fondamentale que toutes les techniques boîte blanche partagent est que l'implémentation entière du logiciel est prise en compte pendant le test, ce qui facilite la détection des défauts même lorsque la spécification du logiciel est vague, obsolète ou incomplète. Une faiblesse associée est que si le logiciel n'implémente pas une ou plusieurs exigences, les tests boîte blanche peuvent ne pas détecter les défauts résultant de ces omissions (Watson 1996).

Les techniques de test boîte blanche peuvent être utilisées dans le cadre de tests statiques (par exemple, lors d'exécution à blanc du code). Elles sont bien adaptées pour revoir le code qui n'est pas encore prêt à être exécuté (Hetzel 1988), ainsi que le pseudocode et d'autres logiques de haut niveau ou des logiques descendantes qui peuvent être modélisées à l'aide d'un graphe de flux de contrôle.

Le fait de n'effectuer que des tests boîte noire ne permet pas de mesurer la couverture réelle du code. Les mesures de couverture boîte blanche fournissent une mesure objective de la couverture et fournissent les informations nécessaires pour permettre la génération de tests supplémentaires afin d'augmenter cette couverture et, par conséquent, d'accroître la confiance dans le code.

4.4. Techniques de tests basés sur l'expérience

Les techniques de test basées sur l'expérience, couramment utilisées et abordées dans les sections suivantes, sont:

- Estimation d'erreurs.
- Test exploratoire.
- Test basé sur des checklists.

4.4.1. Estimation d'erreurs

L'estimation d'erreurs est une technique utilisée pour anticiper l'apparition d'erreurs, de défauts et de défaillances, sur la base des connaissances du testeur, notamment :

- La façon dont l'application a fonctionné dans le passé.
- Les types d'erreurs que les développeurs ont tendance à commettre et les types de défauts qui en résultent.
- Les types de défaillances qui se sont produites dans d'autres applications similaires.

En général, les erreurs, les défauts et les défaillances peuvent être liés à : l'entrée (par exemple, entrée correcte non acceptée, paramètres erronés ou manquants), la sortie (par exemple, format incorrect, résultat erroné), la logique (par exemple, cas manquants, opérateur incorrect), le calcul (par exemple, opérande incorrect, calcul erroné), les interfaces (par exemple, non-concordance des paramètres, types incompatibles) ou encore les données (par exemple, initialisation incorrecte, type incorrect).

Les attaques de fautes constituent une approche méthodique de la mise en œuvre de l'estimation d'erreurs. Cette technique exige du testeur qu'il crée ou acquière une liste d'erreurs, de défauts et de défaillances possibles, et qu'il conçoive des tests qui identifieront les défauts associés aux erreurs, exposeront les défauts ou provoqueront les défaillances. Ces listes peuvent être construites sur la base de l'expérience, des données relatives aux défauts et aux défaillances, ou des connaissances communes sur les raisons des défaillances des logiciels.

Voir (Whittaker 2002, Whittaker 2003, Andrews 2006) pour plus d'informations sur l'estimation d'erreurs et les attaques de faute.

4.4.2. Test exploratoire

Dans le test exploratoire, les tests sont simultanément conçus, exécutés et évalués pendant que le testeur découvre l'objet de test. Les tests sont utilisés pour en apprendre davantage sur l'objet de test, pour l'explorer plus en profondeur avec des tests ciblés et pour créer des tests pour les domaines non testés.

Le test exploratoire est parfois structuré au moyen de test basé sur des sessions. Dans une approche basée sur des sessions, les tests exploratoires sont menés dans un laps de temps défini. Le testeur utilise une charte de test contenant des objectifs de test pour guider le test. La session de test est généralement suivie d'un débriefing qui implique une discussion entre le testeur et les parties prenantes intéressées par les résultats de la session de test. Dans cette approche, les objectifs de test peuvent être traités comme des conditions de test de haut niveau. Les éléments de couverture sont identifiés et exercés pendant la session de test. Le testeur peut utiliser des formulaires de session de test pour documenter les étapes suivies et les découvertes faites.

Les tests exploratoires sont utiles lorsque les spécifications sont peu nombreuses ou inadéquates, ou lorsque les tests sont soumis à une forte pression du temps. Le test exploratoire est également utile pour compléter d'autres techniques de test plus formelles. Les tests exploratoires seront plus efficaces si le testeur est expérimenté, s'il a des connaissances dans le domaine et s'il possède un niveau élevé de compétences essentielles, telles que l'esprit d'analyse, la curiosité et la créativité (voir section 1.5.1).

Les tests exploratoires peuvent intégrer l'utilisation d'autres techniques de test (par exemple, les partitions d'équivalence). De plus amples informations sur les tests exploratoires peuvent être trouvées dans (Kaner 1999, Whittaker 2009, Hendrickson 2013).

4.4.3. Test basé sur des checklists

Dans le test basé sur des checklists, un testeur conçoit, implémente et exécute des tests pour couvrir les conditions de test à partir d'une checklist. Les 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 échecs logiciels. Les checklists ne doivent pas contenir d'éléments qui peuvent être vérifiés automatiquement, d'éléments correspondant davantage à des critères d'entrée ou de sortie, ou d'éléments trop généraux (Brykczynski 1999).

Les éléments de la checklist sont souvent formulés sous la forme d'une question. Il doit être possible de vérifier chaque élément séparément et directement. Ces éléments peuvent se référer à des exigences, à des propriétés d'interface graphique, à des caractéristiques-qualité ou à d'autres formes de conditions de test. Des checklists peuvent être créées pour soutenir différents types de tests, y compris les tests fonctionnels et non fonctionnels (par exemple, 10 heuristiques pour les tests d'utilisabilité (Nielsen 1994)).

Certaines entrées de la checklist peuvent devenir progressivement moins efficaces au fil du temps parce que les développeurs apprendront à éviter de commettre les mêmes erreurs. Il peut également s'avérer nécessaire d'ajouter de nouvelles entrées pour tenir compte des défauts de sévérité élevée nouvellement trouvés. Par conséquent, les checklists doivent être régulièrement mises à jour sur la base de l'analyse des défauts. Il convient toutefois de veiller à ce que la checklist ne devienne pas trop longue (Gawande 2009).

En l'absence de cas de test détaillés, le test basés sur des checklists peut fournir des lignes directrices et un certain degré de cohérence pour les tests. Si les checklists sont de haut niveau, une certaine variation dans les tests réels est probable, résultant d'une couverture potentiellement plus grande mais d'une répétabilité moindre.

4.5. Approches de test basées sur la collaboration

Chacune des techniques susmentionnées (voir les sections 4.2, 4.3, 4.4) a un objectif particulier en ce qui concerne la détection des défauts. Les approches basées sur la collaboration, en revanche, se concentrent également sur l'évitement des défauts par la collaboration et la communication.

4.5.1. Rédaction collaborative de User Stories

Une User Story représente une caractéristique qui sera utile à l'utilisateur ou à l'acheteur d'un système ou d'un logiciel. Les User Stories présentent trois aspects essentiels (Jeffries 2000), appelés ensemble les "3 C" :

- Carte - le support décrivant une User Story (par exemple, une carte d'index, une entrée dans un tableau électronique).
- Conversation - explique comment le logiciel sera utilisé (peut être documentée ou verbale)
- Confirmation - les critères d'acceptation (voir section 4.5.2)

Le format le plus courant d'une User Story est le suivant : "En tant que [rôle], je veux que [objectif à atteindre], afin de pouvoir [valeur métier résultante pour le rôle]", suivi des critères d'acceptation.

La rédaction collaborative de la User Story peut faire appel à des techniques telles que le brainstorming et la cartographie mentale. La collaboration permet à l'équipe d'obtenir une vision commune de ce qui

devrait être livré, en prenant en compte trois perspectives : celle du métier, celle du développement et celle du test.

Les bonnes User Story doivent être : Indépendantes, Négociables, apportant de la Valeur, Estimables, Petites et Testables (INVEST). Si une partie prenante ne sait pas comment tester une User Story, cela peut indiquer que la User Story n'est pas assez claire, qu'elle ne reflète pas quelque chose de valable pour elle, ou que la partie prenante a simplement besoin d'aide pour le test (Wake 2003).

4.5.2. Critères d'acceptation

Les critères d'acceptation d'une User Story sont les conditions que doit remplir une implémentation de cette User Story pour être acceptée par les parties prenantes. De ce point de vue, les critères d'acceptation peuvent être considérés comme les conditions de test qui devraient être vérifiées par les tests. Les critères d'acceptation sont généralement le résultat de la conversation (voir section 4.5.1).

Les critères d'acceptation sont utilisés pour :

- Définir le périmètre de la User Story.
- Obtenir un consensus parmi les parties prenantes.
- Décrire les scénarios positifs et négatifs.
- Servir de base aux tests d'acceptation des utilisateurs (voir section 4.5.3).
- Permettre une planification et une estimation précises.

Il existe plusieurs façons de rédiger des critères d'acceptation pour une User Story. Les deux formats les plus courants sont:

- Orienté-scénario (par exemple, le format Given/When/Then (Etant donné/Lorsque/Alors) utilisé par le BDD, voir section 2.1.3)
- Orienté vers les règles (par exemple, liste de vérification à puces, ou tableau de correspondance entrée-sortie).

La plupart des critères d'acceptation peuvent être documentés dans l'un de ces deux formats. Cependant, l'équipe peut utiliser un autre format, personnalisé, tant que les critères d'acceptation sont bien définis et sans ambiguïté.

4.5.3. Développement piloté par les tests d'acceptation (ATDD)

L'ATDD est une approche pilotée par les tests (voir section 2.1.3). Les cas de test sont créés avant d'implémenter la User Story. Les cas de test sont créés par des membres de l'équipe ayant des perspectives différentes, par exemple des clients, des développeurs et des testeurs (Adzic 2009). Les cas de tests peuvent être exécutés manuellement ou de manière automatisée.

La première étape est un atelier de spécification au cours duquel la User Story et (s'ils n'ont pas encore été définis) ses critères d'acceptation sont analysés, discutés et rédigés par les membres de l'équipe. Les lacunes, les ambiguïtés ou les défauts de la User Story sont résolus au cours de ce processus. L'étape suivante consiste à créer les cas de test. Cela peut être fait par l'équipe dans son ensemble ou par le testeur individuellement. Les cas de test sont basés sur les critères d'acceptation et peuvent être considérés comme des exemples du fonctionnement du logiciel. Ils aideront l'équipe à mettre en œuvre correctement la User Story.

Comme les exemples et les tests sont les mêmes, ces termes sont souvent utilisés de manière interchangeable. Lors de la conception des tests, les techniques de test décrites dans les sections 4.2, 4.3 et 4.4 peuvent être appliquées.

En général, les premiers cas de test sont positifs, confirmant le comportement correct sans exceptions ou conditions d'erreur, et comprennent la séquence d'activités exécutées si tout se passe comme prévu. Une fois les cas de test positifs réalisés, l'équipe doit effectuer des tests négatifs. Enfin, l'équipe doit également couvrir les caractéristiques non fonctionnelles de la qualité (par exemple, efficacité de la performance, utilisabilité). Les cas de test doivent être exprimés d'une manière compréhensible pour les parties prenantes. En règle générale, les cas de test contiennent des phrases en langage naturel comprenant les préconditions nécessaires (le cas échéant), les données d'entrée et les postconditions.

Les cas de test doivent couvrir toutes les caractéristiques de la User Story et ne doivent pas aller au-delà. Cependant, les critères d'acceptation peuvent détailler certains des problèmes décrits dans la User Story. En outre, aucun cas de test ne doit décrire les mêmes caractéristiques de la User Story.

Lorsqu'ils sont définis dans un format pris en charge par un framework d'automatisation des tests, les développeurs peuvent automatiser les cas de test en écrivant le code correspondant au fur et à mesure qu'ils implémentent la caractéristique décrite dans une User Story. Les tests d'acceptation deviennent alors des exigences exécutables.

5. Gestion des activités de test – 335 minutes

Mots clés

gestion des défauts, rapport de défaut, critères d'entrée, critères de sortie, risque produit, risque projet, risque, analyse des risques, évaluation des risques, contrôle des risques, identification des risques, niveau de risque, gestion des risques, atténuation des risques, pilotage des risques, tests basés sur les risques, approche de test, rapport de clôture des tests, contrôle des tests, pilotage des tests, planification des tests, rapport d'avancement des tests, pyramide des tests, quadrants des tests.

Objectifs d'apprentissage pour le chapitre 5:

5.1 Planification des tests

FL-5.1.1 (K2) Donner des exemples de l'objectif et du contenu d'un plan de test

FL-5.1.2 (K1) Reconnaître la valeur ajoutée d'un testeur dans la planification des itérations et des releases

FL-5.1.3 (K2) Comparer et opposer les critères d'entrée et les critères de sortie

FL-5.1.4 (K3) Utiliser des techniques d'estimation pour calculer l'effort de test requis

FL-5.1.5 (K3) Appliquer la priorisation des cas de test

FL-5.1.6 (K1) Rappeler les concepts de la pyramide des tests

FL-5.1.7 (K2) Résumer les quadrants du test et leurs relations avec les niveaux et les types de test

5.2 Gestion des risques

FL-5.2.1 (K1) Identifier le niveau de risque en utilisant la probabilité et l'impact du risque

FL-5.2.2 (K2) Distinguer les risques projet des risques produit

FL-5.2.3 (K2) Expliquer comment l'analyse des risques produit peut influencer la rigueur et l'étendue des tests

FL-5.2.4 (K2) Expliquer les mesures qui peuvent être prises en réponse à l'analyse des risques produit

5.3 Pilotage des tests, contrôle des tests et clôture des tests

FL-5.3.1 (K1) Rappeler des métriques utilisées pour le test

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

FL-5.3.3 (K2) Donner des exemples de la manière de communiquer l'état d'avancement des tests

5.4 Gestion de configuration

FL-5.4.1 (K2) Résumer la manière dont la gestion de configuration soutient les tests

5.5 Gestion des défauts

FL-5.5.1 (K3) Préparer un rapport de défaut

5.1. Planification des tests

5.1.1. Objet et contenu d'un plan de test

Un plan des tests décrit les objectifs, les ressources et les processus d'un projet de test. Un plan de test :

- Documente les moyens et le calendrier pour atteindre les objectifs de test.
- Aide à garantir que les activités de test réalisées répondront aux critères établis.
- Sert de moyen de communication avec les membres de l'équipe et les autres parties prenantes.
- Démontre que le test sera conforme à la politique de test et à la stratégie de test existantes (ou explique pourquoi le test s'en écarte).

La planification des tests guide la réflexion des testeurs et les oblige à faire face aux défis futurs liés aux risques, aux calendriers, aux personnes, aux outils, aux coûts, aux efforts, etc. Le processus de préparation d'un plan de test est un moyen utile de réfléchir aux efforts nécessaires pour atteindre les objectifs du projet de test.

Le contenu typique d'un plan de test comprend:

- Le contexte du test (par exemple, périmètre, objectifs de test, contraintes, base de test)
- Les hypothèses et contraintes du projet de test
- Les parties prenantes (par exemple, rôles, responsabilités, pertinence pour les tests, besoins en matière d'embauche et de formation)
- La communication (par exemple, formes et fréquence de communication, modèles de documentation)
- Le référentiel des risques (par exemple, risques produits, risques projet)
- L'approche de test (par exemple, niveaux de test, types de test, techniques de test, livrables de test, critères d'entrée et de sortie, indépendance du test, métriques à collecter, exigences en matière de données de test, exigences en matière d'environnement de test, écarts par rapport à la politique de test et à la stratégie de test de l'organisation)
- Le budget et le calendrier

De plus amples détails sur le plan de test et son contenu peuvent être trouvés dans le standard ISO/IEC/IEEE 29119-3.

5.1.2. Contribution du testeur à la planification des itérations et des releases

Dans les cycles de vie de développement du logiciel itératifs, il existe généralement deux types de planification : la planification de la release et la planification de l'itération.

La planification de la release prévoit la livraison d'un produit, définit et redéfinit le product backlog, et peut impliquer le découpage de User Stories plus importantes en un ensemble de User Stories plus petites. Elle sert également de base à l'approche de test et au plan de test pour toutes les itérations. Les testeurs impliqués dans la planification de la release participent à la rédaction des User Stories testables et des

critères d'acceptation (voir section 4.5), participent à l'analyse des risques projet et des risques qualité (voir section 5.2), estiment l'effort de test associé aux User Stories (voir section 5.1.4), déterminent l'approche de test, et planifient les tests pour la release.

La planification de l'itération se projette à la fin d'une seule itération et se préoccupe du backlog de l'itération. Les testeurs impliqués dans la planification de l'itération participent à l'analyse détaillée des risques des User Stories, déterminent la testabilité des User Stories, décomposent les User Stories en tâches (en particulier les tâches de test), estiment l'effort de test pour toutes les tâches de test et identifient et affinent les aspects fonctionnels et non fonctionnels de l'objet test.

5.1.3. Critères d'entrée et critères de sortie

Les critères d'entrée définissent les préconditions pour entreprendre une activité donnée. Si les critères d'entrée ne sont pas remplis, il est probable que l'activité se révélera plus difficile, plus longue, plus coûteuse et plus risquée. Les critères de sortie définissent ce qui doit être réalisé pour qu'une activité soit déclarée achevée. Les critères d'entrée et les critères de sortie doivent être définis pour chaque niveau de test et diffèrent en fonction des objectifs du test.

Les critères d'entrée typiques comprennent : la disponibilité des ressources (par exemple, le personnel, les outils, les environnements, les données de test, le budget, le temps), la disponibilité du matériel de test (par exemple, la base de test, les exigences testables, les User Stories, les cas de test), et le niveau de qualité initial d'un objet de test (par exemple, tous les smoke tests ont été passés avec succès).

Les critères de sortie typiques comprennent : les mesures de l'exhaustivité (par exemple, le niveau de couverture atteint, le nombre de défauts non résolus, la densité de défauts, le nombre de cas de test en échec), et les critères de clôture (par exemple, les tests planifiés ont été exécutés, les tests statiques ont été effectués, tous les défauts trouvés ont été signalés, tous les tests de régression ont été automatisés).

L'épuisement du temps ou du budget peut également être considéré comme un critère de sortie valable. Même si les autres critères de sortie ne sont pas satisfaits, il peut être acceptable de mettre fin aux tests dans de telles circonstances, si les parties prenantes ont revu et accepté le risque d'une mise en production sans tests supplémentaires.

Dans le développement logiciel Agile, les critères de sortie sont souvent appelés Definition of Done, définissant les métriques objectives de l'équipe pour un produit livrable. Les critères d'entrée qu'une User Story doit remplir pour démarrer les activités de développement et/ou de test sont appelés Definition of Ready.

5.1.4. Techniques d'estimation

L'estimation de l'effort de test consiste à prévoir la quantité de travail liée à l'effort de test nécessaire pour atteindre les objectifs d'un projet de test. Il est important de préciser aux parties prenantes que l'estimation est basée sur un certain nombre d'hypothèses et qu'elle est toujours sujette à des erreurs. L'estimation des petites tâches est généralement plus précise que celle des grandes. Par conséquent, lors de l'estimation d'une tâche importante, celle-ci peut être décomposée en un ensemble de tâches plus petites qui peuvent à leur tour être estimées.

Dans ce syllabus, quatre techniques d'estimation sont décrites.

Estimation basée sur des ratios. Dans cette technique basée sur des métriques, des chiffres sont collectés à partir de projets antérieurs au sein de l'organisation, ce qui permet de déduire des ratios "standards" pour des projets similaires. Les ratios des projets propres à une organisation (par exemple, à partir de données historiques) sont généralement la meilleure source à utiliser dans le processus d'estimation. Ces ratios standards peuvent ensuite être utilisés pour estimer l'effort de test pour le nouveau projet. Par exemple, si, dans le projet précédent, le rapport entre l'effort de développement et l'effort de test était de 3:2 et que, dans le projet actuel, l'effort de développement devrait être de 600 personnes-jours, l'effort de test peut être estimé à 400 personnes-jours.

Extrapolation. Dans cette technique basée sur les métriques, des mesures sont effectuées le plus tôt possible dans le projet en cours afin de recueillir les données. Lorsque l'on dispose de suffisamment d'observations, l'effort requis pour le reste du travail peut être estimé en extrapolant ces données (généralement en appliquant un modèle mathématique). Cette méthode convient parfaitement aux cycles de vie de développement du logiciel itératifs. Par exemple, l'équipe peut extrapoler l'effort de test de la prochaine itération comme étant la moyenne de l'effort des trois dernières itérations.

Delphi large bande. Dans cette technique itérative, basée sur l'expertise, des experts font des estimations basées sur l'expérience. Chaque expert, de manière isolée, évalue l'effort. Les résultats sont collectés et s'il y a des écarts par rapport aux limites convenues, les experts discutent de leurs estimations actuelles. Chaque expert est alors invité à faire une nouvelle estimation sur la base de ce feedback, toujours de manière isolée. Ce processus est répété jusqu'à ce qu'un consensus soit atteint. Le planning poker est une variante du Delphi large bande, couramment utilisé dans le développement logiciel Agile. Dans le Planning Poker, les estimations sont généralement réalisées à l'aide de cartes sur lesquelles figurent des chiffres représentant l'ampleur de l'effort.

Estimation en trois points. Dans cette technique basée sur l'expertise, trois estimations sont faites par des experts : l'estimation la plus optimiste (a), l'estimation la plus probable (m) et l'estimation la plus pessimiste (b). L'estimation finale (E) est leur moyenne arithmétique pondérée. Dans la version la plus répandue de cette technique, l'estimation est calculée comme suit : E = (a + 4*m + b) / 6. L'avantage de cette technique est qu'elle permet aux experts de calculer l'erreur de mesure : SD = (b - a) / 6. Par exemple, si les estimations (en personnes-heures) sont : a=6, m=9 et b=18, l'estimation finale est de 10±2 personnes-heures (c'est-à-dire entre 8 et 12 personnes-heures), car E = (6 + 4*9 + 18) / 6 = 10 et SD = (18 - 6) / 6 = 2.

Voir (Kan 2003, Koomen 2006, Westfall 2009) pour ces techniques d'estimation de test et bien d'autres.

5.1.5. Priorisation des cas de test

Une fois que les cas de test et les procédures de test sont spécifiés et organisés en suites de tests, ces suites de tests peuvent être organisées dans un calendrier d'exécution des tests qui définit l'ordre dans lequel elles doivent être exécutées. Lors de la priorisation des cas de test, différents facteurs peuvent être pris en compte. Les stratégies de priorisation des cas de test les plus couramment utilisées sont les suivantes:

- Priorisation basée sur les risques, où l'ordre d'exécution des tests est piloté par les résultats de l'analyse des risques (voir section 5.2.3). Les cas de test couvrant les risques les plus importants sont exécutés en premier.
- Priorisation basée sur la couverture, où l'ordre d'exécution des tests est basé sur la couverture (par exemple, la couverture des instructions). Les cas de test ayant la couverture la plus élevée sont exécutés en premier. Dans une autre variante, appelée priorisation de la couverture

additionnelle, le cas de test offrant la couverture la plus élevée est exécuté en premier ; chaque cas de test suivant est celui qui offre la couverture additionnelle la plus élevée.

- Priorisation basée sur les exigences, où l'ordre d'exécution des tests est basé sur les priorités des exigences répercutées sur les cas de test correspondants. Les priorités des exigences sont définies par les parties prenantes. Les cas de test liés aux exigences les plus importantes sont exécutés en premier.

Idéalement, les cas de test devraient être exécutés en fonction de leur niveau de priorité, en utilisant, par exemple, l'une des stratégies de priorisation mentionnées ci-dessus. Cependant, cette pratique peut ne pas fonctionner si les cas de test ou les caractéristiques testées ont des dépendances. Si un cas de test ayant une priorité plus élevée dépend d'un cas de test ayant une priorité moins élevée, le cas de test ayant la priorité la moins élevée doit être exécuté en premier.

L'ordre d'exécution des tests doit également tenir compte de la disponibilité des ressources. Par exemple, les outils de test requis, les environnements de test ou les personnes qui peuvent n'être disponibles que pour une fenêtre de temps spécifique.

5.1.6. Pyramide des tests

La pyramide des tests est un modèle qui montre que différents tests peuvent avoir un niveau de détail différent. Le modèle de la pyramide des tests aide l'équipe dans l'automatisation des tests et dans l'allocation de l'effort de test en montrant que différents objectifs sont soutenus par différents niveaux d'automatisation des tests. Les couches de la pyramide représentent des groupes de tests.

Plus la couche est élevée :

- plus le niveau de détail des tests est faible (les tests sont peu détaillés)
- moins les tests sont isolés (un même test couvre généralement plusieurs fonctionnalités)
- plus le temps d'exécution des tests est important (un test est plus long à exécuter)

Les tests de la couche inférieure sont petits, isolés, rapides et vérifient un petit morceau de fonctionnalité, de sorte qu'il en faut généralement beaucoup pour obtenir une couverture raisonnable. Le niveau supérieur représente les tests complexes, de haut niveau et de bout en bout. Ces tests de haut niveau sont généralement plus lents que les tests des couches inférieures, et ils vérifient généralement un grand nombre de fonctionnalités, de sorte qu'il suffit généralement d'un petit nombre d'entre eux pour obtenir une couverture raisonnable. Le nombre et le nom des couches peuvent varier. Par exemple, le modèle original de pyramide des tests (Cohn 2009) définit trois couches : "tests unitaires", "tests de service" et "tests d'interface utilisateur". Un autre modèle populaire définit des tests unitaires (composants), des tests d'intégration (intégration de composants) et des tests bout en bout. D'autres niveaux de test (voir section 2.2.1) peuvent également être utilisés.

5.1.7. Les quadrants de tests

Les quadrants de tests, définis par Brian Marick (Marick 2003, Crispin 2008), regroupent les niveaux de tests avec les types de tests, les activités, les techniques de tests et les produits d'activités appropriés dans le développement logiciel Agile. Le modèle aide le management des tests à les visualiser pour s'assurer que tous les types et niveaux de test appropriés sont inclus dans le cycle de vie du développement logiciel, et à comprendre que certains types de test sont plus pertinents pour certains

niveaux de test que pour d'autres. Ce modèle permet également de différencier et de décrire les types de tests à toutes les parties prenantes, y compris les développeurs, les testeurs et les représentants métier.

Dans ce modèle, les tests peuvent être orientés métier ou technologie. Les tests peuvent également soutenir l'équipe (c'est-à-dire guider le développement) ou critiquer le produit (c'est-à-dire mesurer son comportement par rapport aux attentes). La combinaison de ces deux points de vue détermine les quatre quadrants:

- Quadrant Q1 (orienté vers la technologie, soutien à l'équipe). Ce quadrant contient les tests de composants et d'intégration de composants. Ces tests doivent être automatisés et inclus dans le processus d'intégration continue.
- Quadrant Q2 (orienté vers le métier, soutien à l'équipe). Ce quadrant contient des tests fonctionnels, des exemples, des tests de User Story, des prototypes d'expérience utilisateur, des tests d'API et des simulations. Ces tests vérifient les critères d'acceptation et peuvent être manuels ou automatisés.
- Quadrant Q3 (orienté vers le métier, critique du produit). Ce quadrant contient les tests exploratoires, les tests d'utilisabilité, les tests d'acceptation utilisateurs. Ces tests sont orientés vers l'utilisateur et souvent manuels.
- Quadrant Q4 (orienté vers la technologie, critique du produit). Ce quadrant contient les smoke tests et les tests non fonctionnels (à l'exception des tests d'utilisabilité). Ces tests sont souvent automatisés.

5.2. Gestion des risques

Les organisations sont confrontées à de nombreux facteurs internes et externes qui rendent incertains la réalisation de leurs objectifs et le moment où ils seront atteints (ISO 31000). La gestion des risques permet aux organisations d'augmenter la probabilité d'atteindre leurs objectifs, d'améliorer la qualité de leurs produits et d'accroître la confiance des parties prenantes.

Les principales activités de gestion des risques sont les suivantes:

- Analyse des risques (comprend l'identification des risques et l'évaluation des risques ; voir le point 5.2.3)
- Contrôle des risques (comprend l'atténuation des risques et le pilotage des risques ; voir section 5.2.4).

L'approche de test, dans laquelle les activités de test sont sélectionnées, priorisées et gérées sur la base de l'analyse des risques et du contrôle des risques, est appelée « test basé sur les risques ».

5.2.1. Définition du risque et attributs du risque

Le risque est un événement potentiel, un danger, une menace ou une situation dont la survenance entraîne un effet négatif. Un risque peut être caractérisé par deux facteurs :

- La **probabilité du risque** - la probabilité que le risque se produise (supérieure à zéro et inférieure à un).

- L'**impact du risque** (préjudice) - les conséquences de cette occurrence.

Ces deux facteurs expriment le niveau de risque, qui est une mesure du risque. Plus le niveau de risque est élevé, plus son traitement est important.

5.2.2. Risques projet et risques produit

Dans le test logiciel, on s'intéresse généralement à deux types de risques : les risques projet et les risques produit.

Les risques projet sont liés à la gestion et au contrôle du projet. Les risques projet comprennent :

- Les problèmes organisationnels (par exemple, des retards dans la livraison des produits d'activités, des estimations inexactes, des réductions de coûts).
- Les problèmes humains (par exemple, compétences insuffisantes, conflits, problèmes de communication, manque de personnel).
- Les problèmes techniques (par exemple, le dépassement du périmètre, le manque de soutien des outils).
- Les problèmes liés aux fournisseurs (par exemple, défaillance de livraison d'un tiers, faillite de l'entreprise de soutien).

Les risques liés au projet, lorsqu'ils se produisent, peuvent avoir un impact sur le calendrier, le budget ou le périmètre du projet, ce qui affecte la capacité du projet à atteindre ses objectifs.

Les risques produit sont liés aux caractéristiques-qualité du produit (par exemple, décrites dans le modèle de qualité ISO 25010). Voici quelques exemples de risques produit : fonctionnalités manquantes ou incorrectes, calculs erronés, erreurs d'exécution, architecture médiocre, algorithmes inefficaces, temps de réponse inadéquat, mauvaise expérience utilisateur, vulnérabilités de sécurité. Les risques produits, lorsqu'ils se produisent, peuvent entraîner diverses conséquences négatives, notamment **:**

- L'insatisfaction des utilisateurs.
- La perte de revenus, de confiance et de réputation.
- Des dommages causés à des tiers.
- Des coûts de maintenance élevés, une surcharge du service d'assistance.
- Des sanctions pénales.
- Dans les cas extrêmes, des dommages physiques, des blessures ou même des décès.

5.2.3. Analyse des risques produits

Du point de vue du test, l'objectif de l'analyse des risques produit est de fournir une conscience du risque produit, afin de concentrer l'effort de test, de manière à minimiser le niveau de risque résiduel du produit. Idéalement, l'analyse des risques produit commence dès le début du cycle de vie du développement logiciel.

L'analyse des risques produit comprend l'identification des risques et l'évaluation des risques. L'identification des risques consiste à dresser une liste exhaustive des risques. Les parties prenantes peuvent identifier les risques à l'aide de diverses techniques et outils, tels que le brainstorming, les ateliers, les entretiens ou les diagrammes de cause à effet.

L'évaluation des risques consiste à catégoriser les risques identifiés, à déterminer leur probabilité, leur impact et leur niveau, à les classer par ordre de priorité et à proposer des moyens de les gérer. La catégorisation facilite l'attribution des mesures d'atténuation, car les risques appartenant à la même catégorie peuvent généralement être atténués par une approche similaire.

L'évaluation des risques peut s'appuyer sur une approche quantitative ou qualitative, ou sur une combinaison des deux. Dans l'approche quantitative, le niveau de risque est calculé en multipliant la probabilité du risque et l'impact du risque. Dans l'approche qualitative, le niveau de risque peut être déterminé à l'aide d'une matrice de risque.

L'analyse des risques produit peut influencer l'exhaustivité et le périmètre des tests. Ses résultats sont utilisés pour:

- Déterminer le périmètre des tests à effectuer.
- Déterminer les niveaux de test particuliers et proposer des types de test à effectuer.
- Déterminer les techniques de test à employer et la couverture à obtenir.
- Estimer l'effort de test requis pour chaque tâche.
- Établir un ordre de priorité pour les tests afin de trouver les défauts critiques le plus tôt possible.
- Déterminer si, outre les tests, d'autres activités pourraient être mises en œuvre pour réduire les risques.

5.2.4. Contrôle des risques produit

Le contrôle des risques produit comprend toutes les mesures prises en réponse aux risques produit identifiés et évalués. Le contrôle des risques produits comprend l'atténuation des risques et la surveillance des risques. L'atténuation des risques consiste à mettre en œuvre les actions proposées lors de l'évaluation des risques, afin de réduire le niveau de risque. La surveillance des risques a pour but de s'assurer que les mesures d'atténuation sont efficaces, d'obtenir des informations supplémentaires pour améliorer l'évaluation des risques et d'identifier les risques émergents.

En ce qui concerne le contrôle des risques produit, une fois qu'un risque a été analysé, plusieurs options de réponse au risque sont possibles, par exemple l'atténuation des risques par des tests, l'acceptation des risques, le transfert des risques ou un plan d'urgence (Veenendaal 2012). Les mesures qui peuvent être prises pour atténuer les risques produit par des tests sont les suivantes:

- Sélectionner les testeurs ayant le bon niveau d'expérience et de compétences, adaptés à un type de risque donné.
- Appliquer un niveau approprié d'indépendance du test.
- Effectuer des revues et réaliser des analyses statiques.
- Appliquer les techniques de test et les niveaux de couverture appropriés.
- Appliquer les types de tests appropriés en fonction des caractéristiques qualité concernées.

- Effectuer des tests dynamiques, y compris des tests de régression.

5.3. Pilotage des tests, contrôle des tests et clôture des tests

Le pilotage des tests concerne la collecte d'informations sur les tests. Ces informations sont utilisées pour évaluer l'avancement des tests et pour mesurer si les critères de sortie des tests ou les tâches de test associées aux critères de sortie sont satisfaits, comme la réalisation des objectifs de couverture des risques, des exigences ou des critères d'acceptation du produit.

Le contrôle des tests utilise les informations issues du pilotage des tests pour fournir, sous la forme de directives de contrôle, des conseils et les actions correctives nécessaires pour réaliser les tests les plus efficaces et les plus efficients. Voici quelques exemples de directives de contrôle:

- Redéfinir l'ordre de priorité des tests lorsqu'un risque identifié devient un problème.
- Réévaluer si un élément de test répond aux critères d'entrée ou de sortie en raison d'un changement.
- Ajuster le calendrier des tests pour tenir compte d'un retard dans la livraison de l'environnement de test.
- Ajouter de nouvelles ressources en cas de besoin.

La clôture des tests recueille des données provenant d'activités de test achevées afin de consolider l'expérience, le testware et toute autre information pertinente. Les activités de clôture des tests interviennent à des étapes clés du projet, par exemple lorsqu'un niveau de test est achevé, qu'une itération Agile est terminée, qu'un projet de test est achevé (ou annulé), qu'un système logiciels est livré, ou qu'une version de maintenance est achevée.

5.3.1. Métriques utilisées pour les tests

Les métriques de test sont recueillies pour montrer l'avancement par rapport au calendrier et au budget prévus, la qualité actuelle de l'objet de test et l'efficacité des activités de test par rapport aux objectifs ou à un but d'itération. Le pilotage des tests rassemble une variété de métriques pour soutenir le contrôle des tests et la clôture des tests.

Les métriques de test les plus courantes sont les suivantes:

- Métriques d'avancement du projet (par exemple, clôture des tâches, utilisation des ressources, effort de test).
- Métriques d'avancement des tests (par exemple, avancement de l'implémentation des cas de test, avancement de la préparation de l'environnement de test, nombre de cas de test exécutés/non exécutés, réussis/échecs, durée d'exécution des tests).
- Métriques de qualité du produit (par exemple, disponibilité, temps de réponse, temps moyen jusqu'à la défaillance).
- Métriques relatives aux défauts (par exemple, nombre et priorités des défauts trouvés/corrigés, densité des défauts, pourcentage de détection des défauts).

- Métriques de risque (par exemple, niveau de risque résiduel).
- Métriques de couverture (par exemple, couverture des exigences, couverture du code).
- Métriques de coût (par exemple, coût du test, coût organisationnel de la qualité).

5.3.2. Objet, contenu et destinataires des rapports de tests

Le reporting des tests résume et communique les informations relatives aux tests pendant et après les tests. Les rapports d'avancement des tests contribuent au contrôle continu des tests et doivent fournir suffisamment d'informations pour permettre de modifier le calendrier des tests, les ressources ou le plan de test, lorsque de telles modifications sont nécessaires en raison d'un écart par rapport au plan ou d'un changement de circonstances. Les rapports de clôture des tests résument une étape spécifique des tests (par exemple, niveau de test, cycle de test, itération) et peuvent fournir des informations pour les tests ultérieurs.

Pendant le pilotage et le contrôle des tests, l'équipe de test génère des rapports d'avancement des tests pour les parties prenantes afin de les tenir informées. Les rapports d'avancement des tests sont généralement générés sur une base régulière (par exemple, chaque jour, chaque semaine, etc.) et comprennent les éléments suivants:

- Période de test.
- Progression des tests (par exemple, avance ou retard par rapport au calendrier), y compris tout écart notable.
- Obstacles aux tests et leurs solutions de contournement.
- Métriques de test (voir section 5.3.1 pour des exemples).
- Risques nouveaux et modifiés au cours de la période de test.
- Tests planifiés pour la prochaine période.

Un rapport de clôture des tests est préparé lors de la clôture des tests, lorsqu'un projet, un niveau de test ou un type de test est terminé et que, dans l'idéal, ses critères de sortie ont été remplis. Ce rapport utilise les rapports d'avancement des tests et d'autres données. Les rapports de clôture des tests les plus courants sont les suivants:

- Résumé du test.
- Évaluation du test et de la qualité du produit sur la base du plan de test initial (c'est-à-dire les objectifs du test et les critères de sortie).
- Écarts par rapport au plan de test (par exemple, différences par rapport au calendrier, à la durée et à l'effort prévus).
- Obstacles au test et solutions de contournement.
- Métriques de test basées sur les rapports d'avancement des tests.
- Risques non atténués, défauts non corrigés.
- Leçons apprises pertinentes pour le test.

Les exigences des différents destinataires de ces rapports sont différentes et influencent leur degré de formalité et leur fréquence. Les rapports sur l'avancement des tests destinés à d'autres membres de la même équipe sont souvent fréquents et informels, tandis que les rapports sur les tests d'un projet clôturé suivent un modèle préétabli et ne sont établis qu'une seule fois.

La norme ISO/IEC/IEEE 29119-3 comprend des modèles et des exemples de rapports d'avancement des tests (appelés rapports d'état des tests) et de rapports de clôture des tests.

5.3.3. Communication de l'état d'avancement des tests

Le meilleur moyen de communiquer l'état d'avancement des tests varie en fonction des préoccupations en matière de gestion des tests, des stratégies de test de l'organisation, des standards réglementaires ou, dans le cas d'équipes auto-organisées (voir section 1.5.2), en fonction de l'équipe elle-même. Les options sont les suivantes :

- Communication verbale avec les membres de l'équipe et les autres parties prenantes.
- Tableaux de bord (par exemple, tableaux de bord intégration continue/développement continu, tableaux des tâches et burn-down charts).
- Canaux de communication électronique (par exemple, mail, chat).
- Documentation en ligne.
- Rapports de tests formels (voir section 5.3.2).

Une ou plusieurs de ces options peuvent être utilisées. Une communication plus formelle peut être plus appropriée pour les équipes distribuées, où la communication directe en face à face n'est pas toujours possible, en raison de la distance géographique ou du décalage horaire. En règle générale, les différentes parties prenantes s'intéressent à différents types d'informations ; la communication doit donc être adaptée en conséquence.

5.4. Gestion de configuration

Dans le domaine du test, la gestion de configuration est une discipline qui permet d'identifier, de contrôler et de suivre les produits d'activités tels que les plans de test, les stratégies de test, les conditions de test, les cas de test, les scripts de test, les résultats de test, les logs de test et les rapports de test ; en tant qu'éléments de configuration.

Pour un élément de configuration complexe (par exemple, un environnement de test), la gestion de configuration enregistre les éléments qui le composent, leurs relations et leurs versions. Si l'élément de configuration est approuvé pour les tests, il devient une base de référence et ne peut être modifié que dans le cadre d'un processus formel de contrôle des modifications.

La gestion de configuration conserve un enregistrement des modifications apportées aux éléments de configuration lorsqu'une nouvelle base de référence est créée. Il est possible de revenir à une base de référence antérieure pour reproduire les résultats des tests précédents.

Pour soutenir correctement les tests, la gestion de configuration garantit les éléments suivants:

- Tous les éléments de configuration, y compris les éléments de test (parties individuelles de l'objet de test), sont identifiés de manière unique, contrôlés par version, suivis pour les modifications et reliés à d'autres éléments de configuration de manière à ce que la traçabilité puisse être maintenue tout au long du processus de test
- Tous les éléments de documentation et de logiciel identifiés sont référencés sans ambiguïté dans la documentation du test

L'intégration continue, la livraison continue, le déploiement continu et les tests associés sont généralement implémentés dans le cadre d'un pipeline DevOps automatisé (voir section 2.1.4), dans lequel la gestion de configuration automatisée est normalement incluse.

5.5. Gestion des défauts

L'un des principaux objectifs des tests étant de trouver des défauts, il est essentiel de mettre en place un processus de gestion des défauts. Bien que nous parlions ici de "défauts", les anomalies signalées peuvent s'avérer être de véritables défauts ou quelque chose d'autre (par exemple, un faux positif, une demande de modification) - ce problème est résolu au cours du processus de traitement des rapports de défauts. Les anomalies peuvent être signalées à n'importe quelle phase du cycle de vie du développement logiciel et leur forme dépend du cycle de vie. Le processus doit être suivi par toutes les parties prenantes concernées. Au minimum, le processus de gestion des défauts comprend un workflow de traitement des anomalies individuelles, de leur découverte à leur clôture, ainsi que des règles de classification. Le workflow comprend généralement des activités visant à enregistrer les anomalies signalées, à les analyser et à les classer, à décider d'une responsabilité appropriée, telle que la correction ou le maintien en l'état, et enfin à clôturer le rapport de défaut. Le processus doit être suivi par toutes les parties prenantes concernées. Il est conseillé de traiter les défauts issus des tests statiques (en particulier l'analyse statique) de la même manière.

Les rapports de défaut typiques ont les objectifs suivants:

- Fournir aux personnes responsables du traitement et de la résolution des défauts signalés des informations suffisantes pour résoudre le problème.
- Fournir un moyen de suivre la qualité du produit d'activités.
- Fournir des idées pour l'amélioration des processus de développement et de test.

Un rapport de défaut enregistré au cours du test dynamique comprend généralement les éléments suivants:

- Identifiant unique
- Titre et bref résumé de l'anomalie signalée
- Date à laquelle l'anomalie a été observée, organisation émettrice et auteur, y compris son rôle.
- Identification de l'objet de test et de l'environnement de test
- Contexte du défaut (par exemple, cas de test en cours d'exécution, activité de test en cours d'exécution, phase du cycle de vie du développement logiciel, et autres informations pertinentes telles que la technique de test, la checklist ou les données de test utilisées)

- Description de la défaillance pour permettre sa reproduction et sa résolution, y compris les étapes qui ont permis de détecter l'anomalie, et tous les logs de test, extraits de base de données, captures d'écran ou enregistrements pertinents.
- Résultats attendus et résultats réels.
- Sévérité du défaut (degré d'impact) sur les intérêts des parties prenantes ou les exigences.
- Priorité de correction.
- Statut du défaut (par exemple, ouvert, différé, dupliqué, en attente de correction, en attente de test de confirmation, ré-ouvert, fermé, rejeté).
- Références (par exemple, au cas de test).

Certaines de ces données peuvent être incluses automatiquement lors de l'utilisation d'outils de gestion des défauts (par exemple, l'identifiant, la date, l'auteur et l'état initial). Des modèles de documents pour un rapport de défaut et des exemples de rapports de défaut peuvent être trouvés dans le standard ISO/IEC/IEEE 29119-3, qui se réfère aux rapports de défaut en tant que rapports d'incident.

6. Outils de test – 20 minutes

Mots clés

automatisation des tests

Objectifs d'apprentissage pour le chapitre 6:

6.1 Support d'outils pour les tests

FL-6.1.1 (K2) Expliquer comment différents types d'outils de test soutiennent les tests

6.2 Avantages et risques de l'automatisation des tests

FL-6.2.1 (K1) Rappeler les avantages et les risques de l'automatisation des tests

6.1. Les outils pour soutenir les tests

Les outils de test soutiennent et facilitent de nombreuses activités de test. Voici quelques exemples:

- Outils de Gestion - augmentent l'efficience du processus de test en facilitant la gestion du cycle de vie du développement logiciel, des exigences, des tests, des défauts et de la configuration.
- Outils de test statique - aident le testeur à effectuer des revues et des analyses statiques.
- Outils de conception et d'implémentation des tests - facilitent la génération des cas de test, des données de test et des procédures de test.
- Outils d'exécution des tests et de couverture - facilitent l'exécution automatisée des tests et la mesure de la couverture.
- Outils de tests non fonctionnels - permettent au testeur d'effectuer des tests non fonctionnels difficiles ou impossibles à réaliser manuellement.
- Outils DevOps - soutiennent le pipeline de livraison DevOps, le suivi du workflow, le(s) processus de build automatisé(s), l'intégration continue/le développement continu.
- Outils de collaboration - facilitent la communication
- Outils prenant en charge l'évolutivité et la standardisation du déploiement (par exemple, machines virtuelles, outils de conteneurisation).
- Tout autre outil aidant aux tests (par exemple, un tableur est un outil de test dans le contexte des tests).

6.2. Avantages et risques de l'automatisation des tests

Le simple fait d'acquérir un outil n'est pas une garantie de succès. Chaque nouvel outil nécessitera des efforts pour obtenir des avantages réels et durables (par exemple, pour l'introduction de l'outil, la maintenance et la formation). Il existe également certains risques, qu'il convient d'analyser et d'atténuer.

Les avantages potentiels de l'automatisation des tests sont les suivants:

- Gain de temps grâce à la réduction du travail manuel répétitif (par exemple, exécution des tests de régression, réintroduction des mêmes données de test, comparaison des résultats attendus et des résultats réels, et vérification des normes de codage).
- Prévention des erreurs humaines simples grâce à une cohérence et une répétabilité accrues (par exemple, les tests sont dérivés de manière cohérente des exigences, les données de test sont créées de manière systématique et les tests sont exécutés par un outil dans le même ordre et à la même fréquence).
- Evaluation plus objective (par exemple, de la couverture) et fourniture de mesures qui sont trop compliquées à calculer pour les humains.
- Accès plus facile à l'information sur les tests pour soutenir la gestion des tests et le reporting des tests (par exemple, statistiques, graphiques et données agrégées sur l'avancement des tests, les taux de défaut et la durée d'exécution des tests).

- Réduction des délais d'exécution des tests pour une détection plus précoce des défauts, un feedback plus rapide et une mise sur le marché plus rapide.
- Plus de temps pour les testeurs pour concevoir de nouveaux tests plus approfondis et plus efficaces.

Les risques potentiels de l'automatisation des tests sont les suivants:

- Attentes irréalistes quant aux avantages d'un outil (y compris ses fonctionnalités et sa facilité d'utilisation).
- Estimations inexactes du temps, des coûts et des exigences liés à l'introduction d'un outil, à la maintenance des scripts de test et à la modification du processus de test manuel existant.
- Utilisation d'un outil de test alors que les tests manuels sont plus appropriés.
- Trop grande dépendance à l'égard d'un outil, par exemple en ignorant la nécessité d'une réflexion humaine critique.
- Dépendance à l'égard du fournisseur de l'outil qui peut faire faillite, retirer l'outil, le vendre à un autre fournisseur ou offrir un support médiocre (par exemple, réponses aux requêtes, mises à jour et corrections de défauts).
- Utilisation d'un logiciel libre qui peut être abandonné, ce qui signifie qu'aucune mise à jour n'est disponible, ou dont les composants internes peuvent nécessiter des mises à jour assez fréquentes dans le cadre d'un développement ultérieur.
- Incompatibilité de l'outil d'automatisation avec la plateforme de développement.
- Choix d'un outil inadapté qui n'a pas respecté les exigences réglementaires et/ou les standards de sécurité.

7. Références

Standards

ISO/IEC/IEEE 29119-1 (2022) Software and systems engineering – Software testing – Part 1: General Concepts

ISO/IEC/IEEE 29119-2 (2021) Software and systems engineering – Software testing – Part 2: Test processes

ISO/IEC/IEEE 29119-3 (2021) Software and systems engineering – Software testing – Part 3: Test documentation

ISO/IEC/IEEE 29119-4 (2021) 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

ISO/IEC/IEEE 14764:2022 – Software engineering – Software life cycle processes – Maintenance

ISO 31000 (2018) Risk management – Principles and guidelines

Livres

Adzic, G. (2009) Bridging the Communication Gap: Specification by Example and Agile Acceptance Testing, Neuri Limited

Ammann, P. and Offutt, J. (2016) Introduction to Software Testing (2e), Cambridge University Press

Andrews, M. and Whittaker, J. (2006) How to Break Web Software: Functional and Security Testing of Web Applications and Web Services, Addison-Wesley Professional

Beck, K. (2003) Test Driven Development: By Example, Addison-Wesley

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

Boehm, B. (1981) Software Engineering Economics, Prentice Hall, Englewood Cliffs, NJ

Buxton, J.N. and Randell B., eds (1970), Software Engineering Techniques. Report on a conference sponsored by the NATO Science Committee, Rome, Italy, 27–31 October 1969, p. 16

Chelimsky, D. et al. (2010) The Rspec Book: Behaviour Driven Development with Rspec, Cucumber, and Friends, The Pragmatic Bookshelf: Raleigh, NC

Cohn, M. (2009) Succeeding with Agile: Software Development Using Scrum, Addison-Wesley

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: A Practical Guide for Testers and Agile Teams, Pearson Education: Boston MA

Forgács, I., and Kovács, A. (2019) Practical Test Design: Selection of traditional and automated test design techniques, BCS, The Chartered Institute for IT

Gawande A. (2009) The Checklist Manifesto: How to Get Things Right, New York, NY: Metropolitan Books

Gärtner, M. (2011), ATDD by Example: A Practical Guide to Acceptance Test-Driven Development, Pearson Education: Boston MA

Gilb, T., Graham, D. (1993) Software Inspection, Addison Wesley

Hendrickson, E. (2013) Explore It!: Reduce Risk and Increase Confidence with Exploratory Testing, The Pragmatic Programmers

Hetzel, B. (1988) The Complete Guide to Software Testing, 2nd ed., John Wiley and Sons

Jeffries, R., Anderson, A., Hendrickson, C. (2000) Extreme Programming Installed, Addison-Wesley Professional

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

Kan, S. (2003) Metrics and Models in Software Quality Engineering, 2nd ed., Addison-Wesley

Kaner, C., Falk, J., and Nguyen, H.Q. (1999) Testing Computer Software, 2nd ed., Wiley

Kaner, C., Bach, J., and Pettichord, B. (2011) Lessons Learned in Software Testing: A Context-Driven Approach, 1st ed., Wiley

Kim, G., Humble, J., Debois, P. and Willis, J. (2016) The DevOps Handbook, Portland, OR

Koomen, T., van der Aalst, L., Broekman, B. and Vroon, M. (2006) TMap Next for result-driven testing, UTN Publishers, The Netherlands

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

O'Regan, G. (2019) Concise Guide to Software Testing, Springer Nature Switzerland

Pressman, R.S. (2019) Software Engineering. A Practitioner's Approach, 9th ed., McGraw Hill

Roman, A. (2018) Thinking-Driven Testing. The Most Reasonable Approach to Quality Control, Springer Nature Switzerland

Van Veenendaal, E (ed.) (2012) Practical Risk-Based Testing, The PRISMA Approach, UTN Publishers: The Netherlands

Watson, A.H., Wallace, D.R. and McCabe, T.J. (1996) Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric, U.S. Dept. of Commerce, Technology Administration, NIST

Westfall, L. (2009) The Certified Software Quality Engineer Handbook, ASQ Quality Press

Whittaker, J. (2002) How to Break Software: A Practical Guide to Testing, Pearson

Whittaker, J. (2009) Exploratory Software Testing: Tips, Tricks, Tours, and Techniques to Guide Test Design, Addison Wesley

Whittaker, J. and Thompson, H. (2003) How to Break Software Security, Addison Wesley

Wiegers, K. (2001) Peer Reviews in Software: A Practical Guide, Addison-Wesley Professional

Articles et pages web

Brykczynski, B. (1999) “A survey of software inspection checklists,” *ACM SIGSOFT Software Engineering Notes, 24(1), pp. 82-89*

Enders, A. (1975) “An Analysis of Errors and Their Causes in System Programs,” *IEEE Transactions on Software Engineering 1(2), pp. 140-149*

Manna, Z., Waldinger, R. (1978) “The logic of computer programming,” *IEEE Transactions on Software Engineering 4(3), pp. 199-229*

Marick, B. (2003) Exploration through Example, http://www.exampler.com/old-blog/2003/08/21.1.html#agile-testing-project-1

Nielsen, J. (1994) “Enhancing the explanatory power of usability heuristics,” *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Celebrating Interdependence, ACM Press, pp. 152–158*

Salman. I. (2016) “Cognitive biases in software quality and testing,” *Proceedings of the 38th International Conference on Software Engineering Companion (ICSE '16), ACM, pp. 823-826.*

Wake, B. (2003) “INVEST in Good Stories, and SMART Tasks,” https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/

8. Annexe A - Objectifs d'apprentissage/Niveau de connaissance

Les 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 associé. Les objectifs d'apprentissage commencent par un verbe d'action correspondant à leur niveau cognitif de connaissance, comme indiqué ci-dessous.

Niveau 1 : Se souvenir (K1) - le candidat se souviendra, reconnaîtra et se rappellera d'un terme ou d'un concept.

Verbes d'action : identifier, rappeler, se souvenir, reconnaître.

Exemples :

- "Identifier les objectifs de test typiques".
- "Rappeler les concepts de la pyramide des tests".
- "Reconnaître comment un testeur ajoute de la valeur à la planification des itérations et des releases".

Niveau 2 : Comprendre (K2) - le candidat peut sélectionner les raisons ou les explications des affirmations liées au sujet, et peut résumer, comparer, classer et donner des exemples pour le concept de test.

Verbes d'action : classer, comparer, contraster, différencier, distinguer, exemplifier, expliquer, donner des exemples, interpréter, résumer.

Exemples:

- "Classer les différentes options pour la rédaction des critères d'acceptation".
- " Comparer les différents rôles dans les tests " (rechercher les similitudes, les différences ou les deux).
- " Faire la distinction entre les risques projet et les risques produit" (permet de différencier les concepts).
- " Donner un exemple de l'objectif et du contenu d'un plan de test".
- "Expliquer l'impact du contexte sur le processus de test".
- " Résumer les activités du processus de revue".

Niveau 3 : Appliquer (K3) - le candidat peut exécuter une procédure lorsqu'il est confronté à une tâche familière, ou sélectionner la procédure correcte et l'appliquer dans un contexte donné.

Verbes d'action : appliquer, mettre en œuvre, préparer, utiliser.

Exemples:

- "Appliquer la priorisation des cas de test" (doit faire référence à une procédure, une technique, un processus, un algorithme, etc.).
- "Préparer un rapport de défaut".
- "Utiliser l'analyse des valeurs limites pour dériver les cas de test".

Références pour les niveaux cognitifs des objectifs d'apprentissage:

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

9. Annexe B - Matrice de traçabilité des objectifs métiers avec les objectifs d'apprentissage

Cette section énumère le nombre d'objectifs d'apprentissage de niveau Fondation liés aux objectifs métier et la traçabilité entre les objectifs métier de niveau Fondation et les objectifs d'apprentissage de niveau Fondation.

Objectifs métier : Niveau Fondation		FL-BO1	FL-BO2	FL-BO3	FL-BO4	FL-BO5	FL-BO6	FL-BO7	FL-BO8	FL-BO9	FL-BO10	FL-BO11	FL-BO12	FL-BO13	FL-BO14
BO1	Comprendre ce qu'est le test et pourquoi il est bénéfique	6													
BO2	Comprendre les concepts fondamentaux du test logiciel		22												
BO3	Identifier l'approche et les activités de test à mettre en œuvre en fonction du contexte du test			6											
BO4	Évaluer et améliorer la qualité de la documentation				9										
BO5	Accroître l'efficacité et l'efficience des tests					20									
BO6	Aligner le processus de test sur le cycle de vie du développement logiciel						6								
BO7	Comprendre les principes de la gestion des tests							6							
BO8	Rédiger et communiquer des rapports de défauts clairs et compréhensibles								1						
BO9	Comprendre les facteurs qui influencent les priorités et les efforts liés aux tests									7					
BO10	Travailler au sein d'une équipe interfonctionnelle										8				
BO11	Connaître les risques et les bénéfices liés à l'automatisation des tests											1			
BO12	Identifier les compétences essentielles requises pour le test												5		
BO13	Comprendre l'impact des risques sur les tests													4	

Objectifs métier : Niveau Fondation		FL-BO1	FL-BO2	FL-BO3	FL-BO4	FL-BO5	FL-BO6	FL-BO7	FL-BO8	FL-BO9	FL-BO10	FL-BO11	FL-BO12	FL-BO13	FL-BO14
BO14	Rendre compte efficacement de l'état d'avancement et de la qualité des tests														4

Chapitre/ section/ sous-section	Objectifs d'apprentissage	Niveau K	Objectifs métiers													
			FL-BO1	FL-BO2	FL-BO3	FL-BO4	FL-BO5	FL-BO6	FL-BO7	FL-BO8	FL-BO9	FL-BO10	FL-BO11	FL-BO12	FL-BO13	FL-BO14
Chapitre 1	**Fondamentaux des tests**															
1,1	**Qu'est-ce que le test ?**															
1.1.1	Identifier les objectifs habituels du test	K1	X													
1.1.2	Faire la différence entre tester et déboguer	K2		X												
1,2	**Pourquoi est-il nécessaire de tester ?**															
1.2.1	Donner des exemples montrant la nécessité des tests	K2	X													
1.2.2	Rappeler la relation entre les tests et assurance qualité	K1		X												
1.2.3	Faire la distinction entre la cause racine, l'erreur, le défaut et la défaillance	K2		X												
1,3	**Principes du test**															
1.3.1	Expliquer les sept principes du test	K2		X												
1,4	**Activités de test, testware et rôles dans le test**															
1.4.1	Résumer les différentes activités et tâches de test	K2			X											
1.4.2	Expliquer l'impact du contexte sur le processus de test	K2			X			X								
1.4.3	Différencier les composants du testware qui soutiennent les activités de test	K2			X											
1.4.4	Expliquer la valeur du maintien de la traçabilité	K2				X	X									

Chapitre/ section/ sous-section	Objectifs d'apprentissage	Niveau K	Objectifs métiers														
			FL-BO1	FL-BO2	FL-BO3	FL-BO4	FL-BO5	FL-BO6	FL-BO7	FL-BO8	FL-BO9	FL-BO10	FL-BO11	FL-BO12	FL-BO13	FL-BO14
1.4.5	Comparer les différents rôles dans le test	K2										X				
1,5	**Compétences essentielles et bonnes pratiques en matière de test**															
1.5.1	Donnez des exemples de compétences génériques requises pour le test	K2												X		
1.5.2	Rappeler les avantages de l'approche équipe intégrée	K1										X				
1.5.3	Distinguer les avantages et les inconvénients de l'indépendance du test	K2			X											
Chapitre 2	**Tester tout au long du cycle de vie du développement logiciel**															
2,1	**Tester dans le contexte d'un cycle de vie du développement logiciel**															
2.1.1	Expliquer l'impact du cycle de vie du développement logiciel choisi sur le test	K2						X								
2.1.2	Rappeler les bonnes pratiques de test qui s'appliquent à tous les cycles de vie du développement logiciel	K1						X								
2.1.3	Rappeler des exemples d'approches de développement piloté par les tests	K1					X									
2.1.4	Résumer la façon dont DevOps pourrait avoir un impact sur le test	K2					X	X			X	X				
2.1.5	Expliquer l'approche shift left	K2					X	X								
2.1.6	Expliquer comment les rétrospectives peuvent être utilisées comme mécanisme d'amélioration des processus.	K2					X					X				
2,2	**Niveaux de test et types de test**															
2.2.1	Distinguer les différents niveaux de test	K2		X	X											
2.2.2	Distinguer les différents types de tests	K2		X												
2.2.3	Distinguer les tests de confirmation des tests de régression	K2		X												
2,3	**Tests de maintenance**															

Chapitre/ section/ sous-section	Objectifs d'apprentissage	Niveau K	Objectifs métiers													
			FL-BO1	FL-BO2	FL-BO3	FL-BO4	FL-BO5	FL-BO6	FL-BO7	FL-BO8	FL-BO9	FL-BO10	FL-BO11	FL-BO12	FL-BO13	FL-BO14
2.3.1	Résumer les tests de maintenance et leurs déclencheurs	K2		X					X							
Chapitre 3	**Test statique**															
3,1	**Bases du test statique**															
3.1.1	Reconnaître les types de produits qui peuvent être examinés par les différentes techniques de test statique.	K1				X	X									
3.1.2	Expliquer la valeur du test statique	K2	X			X	X									
3.1.3	Comparer et opposer les tests statiques et les tests dynamiques.	K2				X	X									
3,2	**Processus de feedback et de revue**															
3.2.1	Identifier les avantages d'un feedback précoce et fréquent de la part des parties prenantes	K1	X			X						X				
3.2.2	Résumer les activités du processus de revue	K2			X	X										
3.2.3	Rappeler quelles sont les responsabilités attribuées aux rôles principaux lors des revues.	K1				X								X		
3.2.4	Comparer et opposer les différents types de revues	K2		X												
3.2.5	Rappeler les facteurs qui contribuent à la réussite d'une revue	K1					X							X		
Chapitre 4	**Analyse et conception des tests**															
4,1	**Aperçu des techniques de test**															
4.1.1	Distinguer les techniques de test boîte noire, boîte blanche et basées sur l'expérience.	K2		X												
4,2	**Techniques de test boîte noire**															
4.2.1	Utiliser les partitions d'équivalence pour dériver les cas de test	K3					X									
4.2.2	Utiliser l'analyse des valeurs limites pour dériver les cas de test	K3					X									

Chapitre/ section/ sous-section	Objectifs d'apprentissage	Niveau K	Objectifs métiers													
			FL-BO1	FL-BO2	FL-BO3	FL-BO4	FL-BO5	FL-BO6	FL-BO7	FL-BO8	FL-BO9	FL-BO10	FL-BO11	FL-BO12	FL-BO13	FL-BO14
4.2.3	Utiliser les tests par tables de décisions pour dériver les cas de test.	K3					X									
4.2.4	Utiliser les tests de transition d'état pour dériver les cas de test.	K3					X									
4,3	**Techniques de test boîte-blanche**															
4.3.1	Expliquer le test des instructions	K2		X												
4.3.2	Expliquer le test des branches	K2		X												
4.3.3	Expliquer la valeur des tests boîte blanche	K2	X	X												
4,4	**Techniques de test basées sur l'expérience**															
4.4.1	Expliquer l'estimation d'erreurs	K2		X												
4.4.2	Expliquer le test exploratoire	K2		X												
4.4.3	Expliquer le test basé sur des checklists	K2		X												
4,5	**Approches de test basées sur la collaboration**															
4.5.1	Expliquer comment rédiger des User Stories en collaboration avec des développeurs et des représentants du métier	K2				X						X				
4.5.2	Classer les différentes options pour la rédaction des critères d'acceptation	K2										X				
4.5.3	Utiliser le développement piloté par les tests d'acceptation (ATDD) pour dériver les cas de test	K3					X									
Chapitre 5	**Gestion des activités de test**															
5,1	**Planification des tests**															
5.1.1	Donner des exemples de l'objectif et du contenu d'un plan de test	K2		X					X							
5.1.2	Reconnaître la valeur ajoutée d'un testeur dans la planification des itérations et des releases	K1	X									X		X		

Chapitre/ section/ sous-section	Objectifs d'apprentissage	Niveau K	Objectifs métiers													
			FL-BO1	FL-BO2	FL-BO3	FL-BO4	FL-BO5	FL-BO6	FL-BO7	FL-BO8	FL-BO9	FL-BO10	FL-BO11	FL-BO12	FL-BO13	FL-BO14
5.1.3	Comparer et opposer les critères d'entrée et les critères de sortie	K2				X		X								X
5.1.4	Utiliser des techniques d'estimation pour calculer l'effort de test requis	K3							X		X					
5.1.5	Appliquer la priorisation des cas de test	K3							X		X					
5.1.6	Rappeler les concepts de la pyramide des tests	K1		X												
5.1.7	Résumer les quadrants du test et leurs relations avec les niveaux et les types de test	K2		X							X					
5,2	**Gestion des risques**															
5.2.1	Identifier le niveau de risque en utilisant la probabilité et l'impact du risque	K1							X						X	
5.2.2	Distinguer les risques projet des risques produit	K2		X											X	
5.2.3	Expliquer comment l'analyse des risques produit peut influencer la rigueur et l'étendue des tests	K2					X				X				X	
5.2.4	Expliquer les mesures qui peuvent être prises en réponse à l'analyse des risques produit	K2		X			X								X	
5,3	**Pilotage des tests, contrôle des tests et clôture des tests**															
5.3.1	Rappeler des métriques utilisées pour le test	K1									X					X
5.3.2	Résumer les objectifs, le contenu et les destinataires des rapports de test	K2					X				X					X
5.3.3	Donner des exemples de la manière de communiquer l'état d'avancement des tests	K2												X		X
5,4	**Gestion de configuration**															
5.4.1	Résumer la manière dont la gestion de configuration soutient les tests	K2					X		X							
5,5	**Gestion des défauts**															

Chapitre/ section/ sous-section	Objectifs d'apprentissage	Niveau K	Objectifs métiers													
			FL-BO1	FL-BO2	FL-BO3	FL-BO4	FL-BO5	FL-BO6	FL-BO7	FL-BO8	FL-BO9	FL-BO10	FL-BO11	FL-BO12	FL-BO13	FL-BO14
5.5.1	Préparer un rapport de défaut	K3		X						X						
Chapitre 6	**Outils de test**															
6,1	**Les outils pour soutenir les tests**															
6.1.1	Expliquer comment différents types d'outils de test soutiennent les tests	K2					X									
6,2	**Avantages et risques de l'automatisation des tests**															
6.2.1	Rappeler les avantages et les risques de l'automatisation des tests	K1					X						X			

10. Annexe C - Notes de livraison ("Release Notes")

ISTQB® Foundation Syllabus v4.0 est une mise à jour majeure basée sur le syllabus de niveau Fondation (v3.1.1) et le syllabus testeur Agile 2014. Pour cette raison, il n'y a pas de notes de livraison détaillées par chapitre et section. Cependant, un résumé des principales modifications est fourni ci-dessous. De plus, dans un document séparé de notes de release, l'ISTQB® fournit une traçabilité entre les objectifs d'apprentissage (LO) de la version 3.1.1 du syllabus niveau Fondation, de la version 2014 du syllabus Testeur Agile, et les objectifs d'apprentissage du nouveau syllabus niveau Fondation v4.0, montrant quels LO ont été ajoutés, mis à jour ou supprimés.

Au moment de la rédaction du syllabus (2022-2023), plus d'un million de personnes dans plus de 100 pays ont passé l'examen de niveau Fondation, et plus de 800 000 sont des testeurs certifiés dans le monde entier. En s'attendant à ce que toutes ces personnes aient lu le syllabus Fondation pour pouvoir passer l'examen, cela fait du syllabus Fondation probablement le document de test logiciel le plus lu de tous les temps ! Cette mise à jour majeure est faite dans le respect de cet héritage et pour améliorer l'opinion de centaines de milliers de personnes supplémentaires sur le niveau de qualité que l'ISTQB® fournit à la communauté mondiale des tests.

Dans cette version, tous les LO ont été modifiés pour les rendre atomiques et pour créer une traçabilité univoque entre les LO et les sections du syllabus, de sorte qu'il n'y a pas de contenu sans LO. L'objectif est de rendre cette version plus facile à lire, à comprendre, à apprendre et à traduire, en se concentrant sur l'amélioration de l'utilité pratique et de l'équilibre entre les connaissances et les compétences.

Cette version majeure a apporté les modifications suivantes:

- Réduction de la taille globale du syllabus. Le syllabus n'est pas un manuel, mais un document qui sert à présenter les éléments de base d'un cours d'introduction aux tests de logiciels, y compris les sujets qui devraient être couverts et à quel niveau. Par conséquent, notamment:
    - Dans la plupart des cas, les exemples sont exclus du texte. Il incombe à l'organisme de formation de fournir les exemples, ainsi que les exercices, au cours de la formation.
    - La "checklist pour la rédaction du syllabus" a été suivie, laquelle suggère la taille maximale du texte pour les objectifs d'apprentissage à chaque niveau K (K1 = max. 10 lignes, K2 = max. 15 lignes, K3 = max. 25 lignes).
- Réduction du nombre de LO par rapport aux syllabus Fondation v3.1.1 et Agile v2014.
    - 14 LO K1 contre 21 LO dans FL v3.1.1 (15) et AT 2014 (6)
    - 42 LO K2 contre 53 LO dans FL v3.1.1 (40) et AT 2014 (13)
    - 8 LO K3 contre 15 LO dans FL v3.1.1 (7) et AT 2014 (8)
- Des références plus étendues à des livres et articles classiques et/ou respectés sur les tests de logiciels et des sujets connexes sont fournies.
- Modifications majeures dans le chapitre 1 (Fondamentaux des tests)
    - La section sur les compétences en matière de test a été élargie et améliorée.
    - Ajout d'une section sur l'approche équipe intégrée (K1)

- La section sur l'indépendance du test a été déplacée du chapitre 5 au chapitre 1

- Modifications majeures dans le chapitre 2 (Les tests tout au long du cycle de vie du développement logiciel)
  - Les sections 2.1.1 et 2.1.2 ont été réécrites et améliorées, les LO correspondants ont été modifiés.
  - Plus d'accent sur des pratiques comme : l'approche pilotée par les tests (K1), shift left (K2), rétrospectives (K2)
  - Nouvelle section sur les tests dans le contexte de DevOps (K2)
  - Le niveau des tests d'intégration est scindé en deux niveaux de test distincts : le test d'intégration de composants et le test d'intégration de systèmes.
- Modifications majeures dans le chapitre 3 (Test statique)
  - La section sur les techniques de revue, ainsi que l'objectif d'apprentissage K3 (appliquer une technique de revue) ont été supprimés.
- Modifications majeures dans le chapitre 4 (Analyse et Conception des tests)
  - Suppression du test des cas d'utilisation (mais toujours présent dans le syllabus d'analyste de test avancé).
  - Plus d'accent sur l'approche du test basée sur la collaboration : nouvel objectif de performance K3 sur l'utilisation de l'ATDD pour dériver des cas de test et deux nouveaux objectifs d'apprentissage K2 sur les User Stories et les critères d'acceptation.
  - Remplacement des tests et de la couverture des décisions par des tests et une couverture des branches (premièrement, la couverture des branches est plus couramment utilisée dans la pratique ; deuxièmement, différents standards définissent la décision différemment, par opposition à la "branche" ; troisièmement, cela résout un défaut subtil, mais sérieux, de l'ancien FL2018 qui prétend que "100% de couverture des décisions implique 100% de couverture des instructions" - cette phrase n'est pas vraie dans le cas de programmes sans décisions).
  - La section sur la valeur des tests boîte blanche a été améliorée.
- Modifications majeures dans le chapitre 5 (Gestion des activités de test).
  - Suppression de la section sur les stratégies et approches de test.
  - Nouveau LO K3 sur les techniques d'estimation de l'effort de test.
  - Plus d'accent sur les concepts et outils bien connus liés à Agile dans la gestion des tests : planification des itérations et des releases (K1), pyramide des tests (K1), et quadrants de tests (K2).
  - La section sur la gestion des risques est mieux structurée en décrivant quatre activités principales : l'identification des risques, l'évaluation des risques, l'atténuation des risques et la surveillance des risques.
- Modifications majeures dans le chapitre 6 (Outils de test)

- Le contenu de certaines questions relatives à l'automatisation des tests a été jugé trop avancé pour le niveau Fondation - la section sur la sélection des outils, la réalisation de projets pilotes et l'introduction d'outils dans l'organisation a été supprimée.

11. Index