Testeur certifié ISTQB®

Syllabus niveau avancé
Analyste Technique de Test

Version 2019

International Software Testing Qualifications Board

Groupe de travail du niveau avancé: Graham Bath (vice-responsable), Rex Black, Judy McKay, Kenji Onoshi, Mike Smith (responsable), Erik van Veenendaal

Traduction française : Olivier DENOO, Eric RIOU du COSQUER et Bruno LEGEARD

Historique des modifications

Version	Date	Remarks
2012	19 Oct 2012	Lancement ISTQB®
2019 V1.0	18 Oct 2019	Lancement ISTQB®
2019 V1.0 FR	01/02/2021	Version française

Table des Matières

Remerciements

Ce document a été produit par une équipe du groupe de travail sur le niveau avancé de l'International Software Testing Qualifications Board: Graham Bath (vice-responsable), Rex Black, Judy McKay, Kenji Onoshi, Mike Smith (responsable), Erik van Veenendaal

Les personnes suivantes ont participé à la revue, aux commentaires et aux choix d'évolutions de ce syllabus :

Dani Almog	Andrew Archer	Rex Black
Armin Born	Sudeep Chatterjee	Tibor Csöndes
Wim Decoutere	Klaudia Dusser-Zieger	Melinda Eckrich-Brájer
Peter Foldhazi	David Frei	Karol Frühauf
Jan Giesen	Attila Gyuri	Matthias Hamburg
Tamás Horváth	N. Khimanand	Jan te Kock
Attila Kovács	Claire Lohr	Rik Marselis
Marton Matyas	Judy McKay	Dénes Medzihradszky
Petr Neugebauer	Ingvar Nordström	Pálma Polyák
Meile Posthuma	Stuart Reid	Lloyd Roden
Adam Roman	Jan Sabak	Péter Sótér
Benjamin Timmermans	Stephanie van Dijck	Paul Weymouth

L'équipe principale remercie l'équipe de revue et les comités nationaux pour leurs suggestions et contributions.

Ce document a été officiellement publié par l'assemblée générale de l'ISTQB® le 18 octobre 2019.

0. Introduction à ce syllabus

0.1 Objectif de ce syllabus

Ce syllabus constitue la base pour le niveau avancé Analyste de Test de l'International Software Testing Qualifications Board. L'ISTQB® fournit ce syllabus :

1. Aux comités nationaux, pour le traduire dans leur langue et pour accréditer des organismes de formation. Les comités nationaux peuvent adapter le syllabus aux besoins particuliers de leur langue et modifier les références pour prendre en compte des publications locales.
2. Aux fournisseurs d'examens pour créer des questions dans leur langue selon les objectifs d'apprentissage de ce syllabus.
3. Aux organismes de formation, pour produire le matériel de formation et déterminer les méthodes d'enseignement adaptées.
4. Aux candidats à la certification, pour se préparer à l'examen (lors d'une formation ou de façon indépendante).
5. A la communauté internationale de l'ingénierie des logiciels et des systèmes pour faire progresser les métiers du test de systèmes et logiciels et pour servir de base à la rédaction de livres et articles.

L'ISTQB® peut permettre à d'autres organismes d'utiliser ce syllabus pour d'autres raisons, à condition qu'ils en fassent préalablement la demande et obtiennent une réponse écrite.

0.2 Le niveau avancé testeur certifié en test logiciel

Le niveau avancé principal est composé de trois syllabus indépendants correspondant aux rôles suivants :

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

La vue générale sur le niveau avancé 2019 de l'ISTQB est un document séparé [ISTQB_AL_OVIEW] qui contient les informations suivantes :

- Bénéfices métier pour chaque syllabus
- Matrice montrant la traçabilité entre les bénéfices métier et les objectifs d'apprentissage
- Résumé de chaque syllabus

0.3 Objectifs d'apprentissage examinables et niveaux de connaissance

Les objectifs d'apprentissage couvrent les bénéfices métier et servent à créer l'examen pour l'obtention de la certification Analyste Technique de Test au niveau Avancé..

Les niveaux de connaissance K2, K3 et K4 des objectifs d'apprentissage sont présentés au début de chaque chapitre et classés de la façon suivante :

- K2: Comprendre
- K3: Appliquer
- K4: Analyser

0.4 Expérience requise

Certains des objectifs d'apprentissage pour l'Analyste Technique de Test supposent une expérience de premier niveau dans les domaines suivants :

- Concepts généraux de programmation
- Concepts généraux en architecture des systèmes

0.5 L'examen Analyste Technique de Test niveau avancé

L'examen de certification Analyste Technique de Test est basé sur ce syllabus. Les réponses aux questions de l'examen peuvent nécessiter d'utiliser des éléments présents dans plusieurs sections de ce syllabus. Chaque section peut donner lieu à des questions, à l'exception de l'introduction et des annexes. Des standards, ouvrages et autres syllabus ISTQB sont inclus comme références, mais leur contenu ne peut pas donner lieu à des questions au-delà de ce qu'il en est résumé dans le syllabus.

Le format de l'examen est un questionnaire à choix multiples contenant 45 questions. Pour réussir l'examen, au moins 65 % du total des points doivent être obtenus.

Les examens peuvent être passés lors d'une formation accréditée ou de façon indépendante (par exemple dans un centre d'examen agréé ou lors d'un examen public). La participation à un cours de formation accrédité n'est pas une condition préalable à l'examen.

0.6 Prérequis au passage de l'examen

Le certificat testeur certifié niveau fondation doit obligatoirement avoir été obtenu avant le passage de l'examen de certification Analyste Technique de Test de niveau avancé.

0.7 Accréditation des formations

Un comité national membre de l'ISTQB peut accréditer des organismes de formation dont le matériel de formation suit ce syllabus. Les organismes de formation peuvent obtenir les consignes d'accréditation auprès du Comité Français des Tests Logiciels ou d'un autre comité pouvant réaliser une accréditation. Un cours accrédité est officiellement reconnu conforme au syllabus et permet à l'organisme de formation d'organiser l'examen ISTQB au moment de la formation.

0.8 Niveau de détail du syllabus

Le niveau de détail de ce syllabus permet la création, au niveau international, de cours et examens cohérents. Pour cela, le syllabus est constitué des éléments suivants :

- Des objectifs généraux de formation décrivant l'intention de l'Analyste Technique de Test de Niveau Avancé
- Une liste d'éléments dont les étudiants doivent être capables de se souvenir.
- Des objectifs d'apprentissage pour chaque domaine de connaissance, décrivant les compétences à obtenir ou des standards.
- Une description des concepts clé, incluant des références vers des sources telles que des écrits reconnus ou des standards

Le contenu du syllabus n'est pas une description exhaustive du domaine de connaissance; il reflète le niveau de détail à couvrir dans les formations du Niveau Avancé. Il se concentre sur les éléments pouvant être appliqués à tout projet logiciel, y compris les projets en mode Agile. Le syllabus ne contient pas d'objectif d'apprentissage spécifique à un cycle de vie de développement logiciel particulier mais il traite la question de la mise en œuvre de ces concepts dans les projets Agiles, dans les projets suivant un autre mode itératif et incrémental, et dans les cycles de vie séquentiels.

0.9 Organisation de ce syllabus

Il y a six chapitres sur lesquels peuvent porter des questions. Le titre principal de chaque chapitre indique le temps minimal nécessaire à la formation et aux exercices pour ce chapitre, les durées ne sont pas indiquées aux niveaux inférieurs. Pour une formation accréditée, le syllabus exige un minimum de 21 heures et 15 minutes d'enseignement, réparties sur les six chapitres comme suit :

- Chapitre 1 : Les tâches de l'Analyste Technique de Test dans le test basé sur les risques (30 minutes)
- Chapitre 2 : Techniques de Test Boîte-Blanche (345 minutes)
- Chapitre 3 : Techniques Analytiques (210 minutes)
- Chapitre 4 : Caractéristiques Qualité pour les Tests Techniques (345 minutes)
- Chapitre 5 : Revues (165 minutes)
- Chapitre 6 : Outils de Test et Automatisation (180 minutes)

1. Les tâches de l'Analyste Technique de Test dans le test basé sur les risques - 30 mins.

Mots clés
risque produit, évaluation des risques, identification des risques, réduction des risques, test basé sur les risques

Objectifs d'apprentissage pour Les tâches de l'Analyste Technique de Test dans le test basé sur les risques

1.2 Tâches du test basé sur les risques

TTA-1.2.1 (K2) Résumer les facteurs de risques génériques que l'Analyste Technique de Test doit généralement prendre en considération
TTA-1.2.2 (K2) Résumer les activités de l'Analyste Technique de Test dans le cadre d'une approche basée sur les risques pour les activités de test

1.1 Introduction

Le Test Manager a la responsabilité globale d'établir et de gérer une stratégie de test basée sur les risques. Le Test Manager demande habituellement la participation de l'Analyste Technique de Test pour s'assurer que l'approche basée sur les risques est mise en œuvre correctement.

Les Analystes Techniques de Test travaillent dans le cadre du test basé sur les risques mis en place par le Test Manager pour le projet. Ils apportent leur connaissance des risques techniques inhérents au projet, tels que les risques liés à la sécurité, à la fiabilité du système et à la performance.

1.2 Tâches du test basé sur les risques

En raison de leur expertise technique particulière, les Analystes Techniques de Test sont activement impliqués dans les tâches suivantes du test basé sur les risques :

- Identification des risques
- Evaluation des risques
- Réduction des risques

Ces tâches sont exécutées de manière itérative tout au long du projet pour faire face aux nouveaux risques produit et à l'évolution des priorités, et pour évaluer et communiquer régulièrement l'état des risques.

1.2.1 Identification des risques

C'est en faisant appel à l'échantillon le plus large possible de parties prenantes que le processus d'identification des risques sera le plus susceptible de détecter le plus grand nombre possible de risques importants. Comme les Analystes Techniques de Test possèdent des compétences techniques uniques, ils sont particulièrement efficaces pour mener des interviews d'experts, faire du brainstroming avec des collègues et analyser les expériences actuelles et passées afin de déterminer où se situent les domaines probables de risque produit. En particulier, les Analystes Techniques de Test travaillent en étroite collaboration avec d'autres parties prenantes, telles que les développeurs, les architectes, les ingénieurs d'exploitation, les Product Owners, les équipes d'assistance locale, et les techniciens du support, pour déterminer les secteurs de risques techniques qui ont un impact sur le produit et le projet. La participation d'autres parties prenantes garantit que tous les points de vue sont pris en considération et cela est généralement géré par le Test Manager.

Les risques qui pourraient être identifiés par l'Analyste Technique de Test sont généralement basés sur les caractéristiques qualité de [ISO25010] énumérées au chapitre 4, et comprennent, par exemple :

- Efficacité de la performance (par ex., incapacité d'atteindre les temps de réponse requis dans des conditions de charge élevées)
- Sécurité (par ex., divulgation de données sensibles par le biais d'attaques de sécurité)
- Fiabilité (par ex., application incapable de répondre à la disponibilité spécifiée dans le contrat de service (SLA : Service Level Agreement))

1.2.2 Evaluation des risques

Alors que l'identification des risques est une question d'identification du plus grand nombre possible de risques pertinents, l'évaluation des risques est l'étude des risques identifiés afin de classer chaque risque et de déterminer la probabilité et l'impact qui lui sont associés. La probabilité d'occurrence est généralement définie comme la probabilité que le problème potentiel puisse exister dans le système sous test.

L'Analyste Technique de Test contribue à la recherche et à la compréhension du potentiel risque produit technique pour chaque élément de risque, tandis que l'Analyste de Test contribue à comprendre l'impact métier du problème s'il se produit.

Les risques projet peuvent avoir une incidence sur le succès global du projet. En règle générale, les risques projet génériques suivants doivent être pris en considération :

- Conflit entre les parties prenantes concernant les exigences techniques
- Problèmes de communication résultant de la distribution géographique de l'organisation de développement
- Outils et technologie (et les compétences utiles associées)
- Pression sur le temps, les ressources et la gestion
- Manque d'assurance qualité antérieure
- Taux de changement élevés des exigences techniques

Les facteurs de risque produit peuvent entraîner un plus grand nombre de défauts. En règle générale, les risques produit générique suivants doivent être considérés :

- Complexité de la technologie
- Complexité de la structure du code
- Quantité de code réutilisé par rapport au nouveau code
- Grand nombre de défauts liés aux caractéristiques qualité techniques (historique des défauts)
- Problèmes d'interfaces techniques et d'intégration

Sur la base de l'information disponible sur les risques, l'Analyste Technique de Test propose un niveau de risque initial conformément aux lignes directrices établies par le Test Manager. Par exemple, le Test Manager peut déterminer que les risques doivent être classés avec une valeur de 1 à 10, le 1 étant le risque le plus élevé. La valeur initiale peut être modifiée par le Test Manager lorsque tous les points de vue des parties prenantes ont été considérés.

1.2.3 Réduction des Risques

Au cours du projet, les Analystes Techniques de Test influencent la façon dont les tests réagissent aux risques. Cela implique généralement les actions suivantes :

- Réduire les risques en exécutant les tests les plus importants (ceux qui s'attaquent aux zones à haut risque) et en mettant en œuvre des mesures d'atténuation et de contingence appropriées, comme indiqué dans le plan de test
- Évaluer des risques à partir d'informations supplémentaires recueillies au fur et à mesure du déroulement du projet, et utiliser cette information pour mettre en œuvre des mesures d'atténuation visant à diminuer la probabilité ou limiter l'impact de ces risques

L' Analyste Technique de Test collabore souvent avec des spécialistes dans des domaines tels que la sécurité et la performance afin de définir des mesures d'atténuation des risques et des éléments de la stratégie de test organisationnel. Des informations supplémentaires peuvent être obtenues avec des Syllabi du niveau Spécialiste de l'ISTQB ®, tel que le syllabus sur les tests de sécurité de niveau Avancé [ISTQB_ALSEC_SYL] et le syllabus sur les tests de performance au niveau Fondation [ISTQB_FLPT_SYL].

2. Techniques de Test Boîte-Blanche - 345 mins.

Mots clés

Test d'API, condition atomique, test de flot de contrôle, complexité cyclomatique, test des décisions, test des conditions/décisions modifiées, test des conditions multiples, test des chemins, court-circuit, test des instructions, technique de test boîte-blanche

Objectifs d'apprentissage pour Techniques de Test Boîte-Blanche

Note : LOs 2.2.1, 2.3.1, 2.4.1, 2.5.1, et 2.6.1 font reference à un "élément de spécification". Cela inclut des éléments tels que des sections de code, des exigences, des User Stories, des cas d'utilisation, et des spécifications fonctionnelles.

2.2 Test des Instructions

TTA-2.2.1 (K3) Ecrire des cas de test pour un élément de spécification donné en appliquant la technique de test des instructions pour atteindre un niveau défini de couverture

2.3 Test des Décisions

TTA-2.3.1 (K3) Ecrire des cas de test pour un élément de spécification donné en appliquant la technique de test des décisions pour atteindre un niveau défini de couverture

2.4 Test des Conditions/Décisions Modifiées

TTA-2.4.1 (K3) Ecrire des cas de test en appliquant la technique de test des conditions/décisions modifiées pour atteindre un niveau défini de couverture

2.5 Test des Conditions Multiples

TTA-2.5.1 (K3) Rédiger des cas de test pour un élément de spécification donné en appliquant la technique de test des conditions multiple pour atteindre un niveau défini de couverture

2.6 Test des Chemins

TTA-2.6.1 (K3) Ecrire des cas de test pour un élément de spécification donné en appliquant la méthode de référence simplifiée de McCabe

2.7 Test d'API

TTA-2.7.1 (K2) Comprendre l'applicabilité du test d'API et les types de défauts qu'il trouve

2.8 Sélectionner une Technique de Test Boîte-Blanche

TTA-2.8.1 (K4) Sélectionnez une technique de test boîte-blanche appropriée selon une situation de projet donnée

2.1 Introduction

Ce chapitre décrit principalement les techniques de test boîte-blanche. Ces techniques s'appliquent au code et à d'autres structures, telles que les diagrammes de flux de processus métier.

Chaque technique spécifique permet d'obtenir systématiquement des cas de test et se concentre sur un aspect particulier de la structure à considérer. Les techniques fournissent des critères de couverture qui doivent être mesurés et associés à un objectif défini par chaque projet ou organisation. Atteindre une couverture complète ne signifie pas que l'ensemble des tests est terminé, mais plutôt que la technique utilisée ne suggère plus de tests utiles pour la structure à l'étude.

Les techniques suivantes sont prises en compte dans ce syllabus :

- Test des instructions
- Test des décisions
- Test des Conditions/Décisions Modifiées
- Test des conditions multiples
- Test des chemins
- Test d'API

La Syllabus Fondation [ISTQB_FL_SYL] introduit le test des instructions et le test des décisions. Le test des instructions exécute les instructions exécutables dans le code, alors que le test des décisions exécute les décisions dans le code et teste le code qui est exécuté en fonction des résultats de la décision.

Les techniques de test des Conditions/Décisions Modifiées et de test des conditions multiples listées ci-dessus sont basées sur des prédicats de décision et trouvent globalement le même type de défauts. Quelle que soit la complexité d'une décision, elle sera évaluées à VRAIE ou FAUSSE, ce qui déterminera le chemin pris à travers le code. Un défaut est détecté lorsque le chemin prévu n'est pas pris parce qu'un prédicat de décision n'a pas été évalué comme prévu.

Les quatre premières techniques sont de plus en plus approfondies (le test des chemins de base est plus complet que le test des instructions et des décisions); des techniques plus approfondies nécessitent généralement la définition de plus de tests afin d'atteindre la couverture prévue et de trouver des défauts plus subtiles.

Se référer à [Bath14], [Beizer90], [Beizer95], [Copeland03], [McCabe96], [ISO29119] et [Koomen06].

2.2 Test des Instructions

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

Applicabilité
Ce niveau de couverture devrait être considéré comme un minimum pour tout code testé.

Limitations/Difficultés
Les décisions ne sont pas prises en considération. Même des pourcentages élevés de couverture des instructions peuvent ne pas détecter certains défauts dans la logique du code.

2.3 Test des décisions

Le test des décisions exécute les décisions dans le code et teste le code qui est exécuté en fonction des résultats de la décision. Pour ce faire, les cas de test suivent les flots de contrôle qui se produisent à partir d'un point de décision (p. ex, pour une instruction IF, un pour la sortie VRAI et un pour la sortie FAUX; pour une instruction CASE; des cas de test seraient requis pour toutes les sorties possibles y compris la sortie par défaut)

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

Comparé aux techniques de test des Conditions/Décisions Modifiées et de test des conditions multiples décrites ci-dessous, le test des décisions prend en compte la décision complète comme un tout et évalue les sorties VRAI et FAUX dans des cas de test séparés.

Applicabilité
Ce niveau de couverture doit être pris en compte lorsque le code testé est à un niveau important de risque, voire critique (see the table in Section 2.8).

Limitations/Difficulties
Étant donné que cela peut exiger plus de cas de test que de tests uniquement au niveau des instructions, cela peut être problématique lorsque le temps d'exécution des tests est un problème. Le test des décisions ne tient pas compte des détails sur la façon dont une décision avec de multiples conditions est prise et peut ne pas détecter les défauts causés par des combinaisons de ces conditions.

2.4 Test des Conditions/Décisions Modifiées

Par rapport au test des décisions, qui examine la décision complète comme un tout et évalue les résultats VRAI et FAUX dans des cas de test distincts, le test des Conditions/Décisions Modifiées examine la façon dont une décision est prise lorsqu'elle comporte plusieurs conditions (sinon il s'agit simplement du test des décisions).

Chaque prédicat de décision est composé d'une ou plusieurs conditions atomiques simples, chacune d'elles étant évaluée comme une valeur bouléenne simple. Celles-ci sont logiquement combinées pour déterminer le résultat final de la décision. Cette technique vérifie que chacune des conditions atomiques affecte indépendamment et correctement le résultat de la décision globale.

Cette technique offre un niveau de couverture plus élevé que la couverture des instructions et des décisions lorsqu'il y a des décisions contenant des conditions multiples.. Avec N conditions atomiques uniques et indépendantes, la couverture des Conditions/Décisions Modifiées peut généralement être atteinte avec N+1 cas de test uniques. La couverture des Conditions/Décisions modifiées nécessite un ensemble de cas de test pour lesquels une condition atomique simple peut déterminer le résultat de la décision de façon indépendante.

Dans l'exemple suivant, une instruction « Si (A ou B) et C, alors ... » est considéré.

	A	B	C	(A ou B) et C
Test 1	VRAI	FAUX	VRAI	VRAI
Test 2	FAUX	VRAI	VRAI	VRAI
Test 3	FAUX	FAUX	VRAI	FAUX
Test 4	VRAI	FAUX	FAUX	FAUX

Dans le test 1, A est VRAI et le résultat global est VRAI. Si A est changé en FAUX (comme dans le test 3, en maintenant les autres valeurs inchangées), le résultat passe à FAUX, montrant ainsi que A peut affecter indépendamment le résultat de la décision.

Dans le test 2, B est VRAI et le résultat global est VRAI. Si B est changé en FAUX (comme dans le test 3, en maintenant les autres valeurs inchangées), le résultat passe à FAUX, montrant ainsi que B peut affecter indépendamment le résultat de la décision.

Dans le test 1, C est VRAI et le résultat global est VRAI. Si C est changé en FAUX (comme dans le test 4, en maintenant d'autres valeurs inchangées), le résultat change à FAUX, montrant ainsi que C peut affecter indépendamment le résultat de la décision.

Notez que contrairement aux techniques de test des instructions et des décisions, il n'y a pas de « niveaux définis de couverture » le test des Conditions/Décisions Modifiées; la couverture est atteinte (càd. 100 %) ou pas.

Applicabilité
Cette technique est utilisée dans l'industrie aérospatiale et d'autres industries pour les systèmes à sécurité critique. Elle est utilisée lors du test d'un logiciel dans lequel une défaillance pourrait provoquer une catastrophe.

Limitations/Difficulties
La réalisation de la couverture des Conditions/Décisions Modifiées peut être compliquée lorsqu'il y a plusieurs occurrences d'une variable dans une décision avec des conditions multiples ; lorsque cela se produit, les conditions peuvent être « couplées ». Selon la décision, il peut ne pas être possible de changer la valeur d'une condition de telle sorte qu'elle seule entraîne le changement du résultat de la décision. L'une des approches pour résoudre ce problème consiste à préciser que seules les conditions atomiques non couplées doivent être testées au niveau des Conditions/Décisions Modifiées. L'autre approche consiste à analyser chaque décision dans laquelle le couplage se fait au cas par cas.

Certains langages de programmation et/ou interpréteurs sont conçus de telle sorte qu'ils mettent en évidence des comportement de court-circuit lors de l'évaluation d'une décision complexe dans le code. C'est-à-dire que le code exécuté peut ne pas évaluer une expression entière si le résultat final de l'évaluation peut être déterminé après avoir évalué seulement une partie de l'expression. Par exemple, si l'évaluation de la décision «A et B»,il n'y a aucune raison d'évaluer B si A a déjà été évalué comme FAUX. Aucune valeur de B ne peut modifier le résultat final de sorte que le code peut gagner du temps d'exécution en n'évaluant pas B. Le court-circuit peut affecter la capacité d'atteindre la couverture MC/DC puisque certains tests requis peuvent ne pas être réalisables.

2.5 Test des Conditions Multiples

Dans de rares cas, il peut être nécessaire de tester toutes les combinaisons possibles de conditions atomiques qu'une décision peut contenir. Ce niveau exhaustif de tests est appelé test des conditions multiples. Le nombre de test requis dépend du nombre de conditions atomiques dans l'énoncé de décision et peut être déterminé en calculant 2^N avec N le nombre de conditions atomiques non couplées. En utilisant le même exemple que précédemment, les tests suivants sont nécessaires pour obtenir une couverture des conditions multiples :

	A	**B**	**C**	**(A ou B) et C**
Test 1	VRAI	VRAI	VRAI	VRAI
Test 2	VRAI	VRAI	FAUX	FAUX

Test 3	VRAI	FAUX	VRAI	VRAI
Test 4	VRAI	FAUX	FAUX	FAUX
Test 5	FAUX	VRAI	VRAI	VRAI
Test 6	FAUX	VRAI	FAUX	FAUX
Test 7	FAUX	FAUX	VRAI	FAUX
Test 8	FAUX	FAUX	FAUX	FAUX

La couverture est mesurée comme le nombre de combinaisons de conditions uniques exécutées par les tests divisé par le nombre total de combinaisons de condition dans l'objet de test, normalement exprimée en pourcentage.

Applicabilité
Cette technique est utilisée pour tester du logiciel embarqué qui devrait fonctionner de manière fiable sans défaillance pendant de longues périodes de temps (p. ex., les commutateurs téléphoniques qui devraient durer 30 ans).

Limitations/Difficultés
Étant donné que le nombre de cas de test peut être tiré directement d'une table de vérité contenant toutes les conditions atomiques, ce niveau de couverture peut facilement être déterminé. Toutefois, le grand nombre de cas de test requis rend la couverture des Conditions/Décisions Modifiées plus réalisable dans la plupart des situations.

Si le langage de programmation utilise le court-circuit, le nombre de cas de test sera souvent réduit, selon l'ordre et le regroupement des opérations logiques qui sont réalisées sur les conditions atomiques.

2.6 Test des Chemins

Le test des chemins consiste en général à identifier des chemins à travers le code, puis à créer des tests pour les couvrir. En théorie, il serait utile de tester chaque chemin unique à travers le système. Dans tout système non trivial, cependant, le nombre de cas de test pourrait devenir excessivement important en raison de la nature des structures de boucle. En revanche, les tests des chemins de base peuvent être effectués de façon réaliste en suivant la « méthode de référence simplifiée » ou "Simplified Baseline Method" développée par McCabe [McCabe96].

La technique est appliquée avec les étapes suivantes :

1. Créer un graphe de flot de contrôle pour un élément de spécification donné (p. ex., code ou spécification de conception fonctionnelle). Notez qu'il peut s'agir également de la première étape de l'analyse du flot de contrôle (voir la section 3.2.1).
2. Sélectionnez un chemin de référence à travers le code (pas un chemin d'exception). Ce chemin de référence devrait être le chemin le plus important à tester – le risque peut être utilisé pour l'identifier.
3. Générer le deuxième chemin en changeant le résultat de la première décision sur le chemin de référence, tout en gardant le nombre maximum de résultats de décision identiques à ceux du chemin de référence.
4. Générer le troisième chemin en commençant à nouveau par le chemin de référence et en changeant le résultat de la deuxième décision sur le chemin. Lorsque des décisions à plusieurs voies sont prises (p. ex., une instruction « Case »), chaque résultat de la décision doit être exercé avant de passer à la décision suivante.

5. Générer d'autres chemins en changeant chacun des résultats sur le chemin de référence. Lorsque de nouvelles décisions sont prises, le résultat le plus important doit être suivi en premier.
6. Une fois que tous les résultats des décisions sur le chemin de référence ont été couverts, appliquez la même approche aux chemins subséquents jusqu'à ce que tous les résultats des décisions dans l'élément de spécification aient été exercés.

Applicabilité
La « méthode de référence simplifiée » ou "Simplified Baseline Method"—telle que définie ci-dessus—est souvent effectuée sur des logiciels critiques. Elle est un bon complément aux autres méthodes couvertes dans ce chapitre parce qu'elle considère les chemins à travers le logiciel plutôt que juste les résultats de décisions.

Limitations/Difficultées
Lorsque le syllabus a été publié, la prise en charge par des outils des tests des chemins de référence était limitée.

Couverture
La technique décrite ci-dessus devrait assurer une couverture complète de tous les chemins linéairement indépendants et le nombre de chemins doit correspondre à la complexité cyclomatique du code. Selon la complexité du code, il peut être utile d'utiliser un outil pour vérifier que la couverture complète de l'ensemble des chemins de référence a été réalisée. La couverture est mesurée comme le nombre de chemins linéairement indépendants exécutés par les tests divisé par le nombre total de chemins linéairement indépendants dans l'objet de test, et est normalement exprimée en pourcentage. Le test des chemins de référence offre des tests plus approfondis que la couverture des décisions, avec une augmentation relativement faible du nombre de tests [NIST96].

2.7 Test d'API

Une interface de programmation d'application (ou API pour Application Programming Interface) est un code qui permet la communication entre différents processus, programmes et/ou systèmes. Les APIs sont souvent utilisées dans une relation client/serveur où un processus fournit une sorte de fonctionnalité à d'autres processus.

Le test d'API est un type de test plutôt qu'une technique. À certains égards, le test d'API est assez similaire au test d'une interface utilisateur graphique (ou GUI pour Graphical User Interface). L'accent est mis sur l'évaluation des valeurs d'entrée et des données retournées.

Les tests négatifs sont souvent cruciaux lorsqu'il s'agit d'APIs. Les programmeurs qui utilisent les APIs pour accéder à des services externes à leur propre code peuvent essayer d'utiliser les APIs d'une façon pour laquelle elles n'étaient pas destinées. Cela signifie qu'une gestion robuste des erreurs est essentielle pour éviter un mauvais fonctionnement. Des tests combinatoires de nombreuses interfaces différentes peuvent être nécessaires parce que les APIs sont souvent utilisées conjointement avec d'autres APIs, et parce qu'une interface unique peut contenir plusieurs paramètres, dont les valeurs peuvent être combinées de plusieurs façons.

Les API sont souvent mal interfaçées, ce qui présente un risque réel de transactions perdues ou de problèmes de synchronisation. Cela nécessite des tests approfondis des mécanismes de récupération et de retest. Une organisation qui fournit une interface API doit s'assurer que tous les services ont une très grande disponibilité, ce qui nécessite souvent des tests de fiabilité stricts par l'éditeur de l'API ainsi que le soutien de l'infrastructure.

Applicabilité

Les tests APIs sont particulièrement importants pour tester les systèmes de systèmes lorsque les systèmes individuels sont distribués ou utilisent le traitement à distance comme un moyen de décharger certaines tâches vers d'autres processeurs. Par exemple :

- Appels de systèmes d'exploitation
- Architectures orientées services: Service-oriented architectures (SOA)
- Appels de procédure à distance (ou RPC puor Remote Procedure Call)
- Services Web (ou Web services)

La conteneurisation logicielle (ou Software containerization [Burns18]) entraîne la division d'un logiciel en plusieurs conteneurs qui communiquent entre eux à l'aide de mécanismes tels que ceux énumérés ci-dessus. Les tests d'API devraient également cibler ces interfaces.

Limitations/Difficultés
Le test d'API nécessite généralement qu'un Analyste Technique de Test technique utilise des outils spécialisés. Étant donné qu'il n'existe généralement pas d'interface graphique directe associée à une API, des outils peuvent être nécessaires pour configurer l'environnement initial, rassembler les données, invoquer l'API et déterminer le résultat.

Couverture
Le test API est une description d'un type de test; il ne dénote aucun niveau spécifique de couverture. Au minimum, le test API devrait inclure des appels vers l'API avec des valeurs d'entrée réalistes et des entrées inattendues pour vérifier le traitement des exceptions. Des tests API plus approfondis peuvent garantir que les entités pouvant être appelées sont exercées au moins une fois ou que tous les appels possibles sont effectués au moins une fois.

Types de défauts
Les types de défauts que l'on peut trouver en testant les APIs sont assez disparates. Les problèmes d'interface sont courants, tout comme les problèmes de traitement des données, les problèmes de synchronisation, la perte de transactions et la duplication de transactions.

2.8 Sélectionner une Technique de Test Boîte-Blanche

Le contexte du système sous test aura un impact sur les niveaux de risque produit et de criticité (voir ci-dessous). Ces facteurs influencent le niveau de couverture requis (et donc la technique de test boîte blanche à utiliser) et la profondeur de couverture à atteindre. En général, plus le système est critique et plus le niveau de risque produit est élevé, plus les exigences en matière de couverture sont rigoureuses et plus il faut de temps et de ressources pour atteindre la couverture souhaitée.

Parfois, le niveau de couverture requis peut être issu des normes qui s'appliquent au système logiciel. Par exemple, si le logiciel doit être utilisé dans un environnement aéronautique, il peut être tenu de se conformer à la norme DO-178C (en Europe, ED-12C.). Cette norme contient les cinq conditions de défaillance suivantes :

A. Catastrophique : conditions de défaillance susceptibles d'empêcher la poursuite en toute sécurité d'un vol et d'un atterrissage.
B. Dangereuse : une défaillance peut avoir un impact négatif important sur la sécurité ou l'efficacité du rendement
C. Majeure : une défaillance est significative, mais moins grave que pour les cas A ou B
D. Mineure : une défaillance est perceptible, mais avec moins d'impact que pour le cas C
E. Sans effet : la défaillance n'a aucun impact sur la sécurité

Si le système logiciel est classé niveau A, il doit être testé avec une couverture de 100% des Conditions/Décisions Modifiées. Si il est classé niveau B, il doit être testé avec une couverture de 100% des décisions et la couverture des Conditions/Décisions modifiées est facultative. Le niveau C requiert au minimum un niveau de 100% de couverture des instructions.

De même, le standard IEC 61508 [IEC61508] est une norme internationale pour la sécurité fonctionnelle des systèmes programmables, électroniques et liés à la sécurité. Cette norme a été adaptée dans de nombreux domaines différents, y compris l'automobile, le ferroviaire, la fabrication, les centrales nucléaires et les machines. La criticité est définie à l'aide d'une échelle de niveau d'intégrité de sécurité (SIL = System Integrity Levels) où SIL1 est le moins critique et SIL4 le plus critique. La norme formule des recommandations pour la couverture par les tests comme le montre le tableau suivant (notez que les définitions exactes pour chaque niveau d'intégrité de sécurité et pour le sens de « recommandé » et « fortement recommandé » sont définies dans la norme).

SIL	100% de couverture des instructions	100% de couverture des branches (decisions)	100% de couverture des Conditions/Décisions modifiées
1	Recommandé	Recommandé	Recommandé
2	**Fortement recommandé**	Recommandé	Recommandé
3	**Fortement recommandé**	**Fortement recommandé**	Recommandé
4	**Fortement recommandé**	**Fortement recommandé**	**Fortement recommandé**

Dans les systèmes modernes, il est rare que tout le traitement se fasse sur un seul système. Les tests d'API doivent être institués chaque fois qu'une partie du traitement sera effectuée à distance. La criticité du système devrait déterminer combien d'effort devraient être investi dans les tests d'API.

3. Techniques Analytiques - 210 mins.

Mots clés

analyse du flot de contrôle, complexité cyclomatique, analyse du flot de données, paire définition-utilisation, analyse dynamique, fuite mémoire, test d'intégration par paires, test d'intégration par voisinage, analyse statique, pointeur sauvage

Objectifs d'apprentissage pour Techniques Analytiques

3.2 Analyse statique

TTA-3.2.1 (K3) Utilisez l'analyse du flot de contrôle pour détecter si le code a des anomalies de flot de contrôle

TTA-3.2.2 (K2) Expliquez comment l'analyse du flot de données est utilisée pour détecter si le code a des anomalies de flot de données

TTA-3.2.3 (K3) Proposer des moyens d'améliorer la maintenabilité du code en appliquant l'analyse statique

TTA-3.2.4 (K2) Expliquer l'utilisation de graphes d'appel pour l'établissement de stratégies de test d'intégration

3.3 Analyse dynamique

TTA-3.3.1 (K3) Appliquer l'analyse dynamique pour atteindre un objectif précis

3.1 Introduction

Il existe deux types d'analyse : l'analyse statique et l'analyse dynamique.

L'analyse statique (section 3.2) englobe les tests analytiques qui peuvent se produire sans exécuter le logiciel. Étant donné que le logiciel n'est pas exécuté, il est examiné soit par un outil, soit par une personne pour déterminer s'il se comportera correctement lorsqu'il sera exécuté. Cette vue statique du logiciel permet une analyse détaillée sans avoir à créer les données et les préconditions préalables qui permettraient à un scénario de s'exécuter.

Notez que les différentes formes de revues pertinentes pour l'Analyste Technique de Test sont couvertes au chapitre 5.

L'analyse dynamique (section 3.3) nécessite l'exécution réelle du code et est utilisée pour trouver des défauts qui sont plus facilement détectés lorsque le code est exécuté (par exemple, des fuites mémoire). L'analyse dynamique, comme l'analyse statique, peut s'appuyer sur des outils ou peut s'appuyer sur une surveillance individuelle du système exécuté selon des indicateurs tels qu'une augmentation rapide de l'utilisation de la mémoire.

3.2 Analyse statique

L'objectif de l'analyse statique est de détecter les défauts réels ou potentiels dans le code et dans l'architecture du système et d'améliorer leur maintenabilité. L'analyse statique est généralement soutenue par des outils.

3.2.1 Analyse du flot de contrôle

L'analyse du flot de contrôle est la technique statique où les étapes suivies dans le cadre d'un programme sont analysées, soit avec un graphe de flot de contrôle, soit avec un outil. Il y a un certain nombre d'anomalies que l'on peut trouver dans un système en utilisant cette technique, par exemple des boucles mal conçues (p. ex., avoir plusieurs points d'entrée), des cibles ambiguës d'appels de fonctions dans certaines langages (p. ex., Scheme), un séquençage incorrect des opérations, etc..

L'analyse du flot de contrôle peut être utilisée pour déterminer la complexité cyclomatique. La valeur de complexité cyclomatique est un entier positif qui représente le nombre de chemins indépendants dans un graphe fortement connecté. Les boucles et les itérations sont ignorées dès lors qu'elles ont été traversées une fois. Chaque chemin, de l'entrée à la sortie, représente un chemin unique à travers le module. Chaque chemin unique doit être testé.

La valeur de la complexité cyclomatique est généralement utilisée pour comprendre la complexité globale d'un module. La théorie de Thomas McCabe [McCabe 76] est que plus le système est complexe, plus il est difficile à maintenir et plus il peut contenir de défauts. De nombreuses études menées au fil des ans ont mis en évidence cette corrélation entre la complexité et le nombre de défauts. Le NIST (National Institute of Standards and Technology) recommande une valeur maximale de complexité de 10. Tout module mesuré avec une plus grande complexité doit être revu pour une éventuelle division en plusieurs modules.

3.2.2 Analyse du flot de données

L'analyse du flot de données couvre une variété de techniques qui recueillent des informations sur l'utilisation de variables dans un système. Le cycle de vie de chaque variable est étudié, (c.-à-d., où elle est déclarée, définie, lue, évaluée et détruite), puisque des anomalies peuvent se produire au cours de l'une de ces opérations ou si les opérations ne se font pas dans le bon ordre.

Une technique courante est appelée notation définition-utilisation où le cycle de vie de chaque variable est divisé en trois actions atomiques différentes :

- d: lorsque la variable est déclarée, définie ou initialisée
- u: lorsque la variable est utilisée ou lue dans un calcul ou une décision
- k: lorsque la variable est tuée (killed), détruite ou hors de portée

Une notation alternative commune pour d-u-k est: d (définir) - r (référencer ou lire) - u (undefine = libérer).

Ces trois actions atomiques sont combinées en paires (« paires définition-utilisation ») pour illustrer le flot de données. Par exemple, un chemin « du » représente un fragment du code où une variable est définie puis utilisée.

On peut citer comme anomalies possibles du flot de données l'exécution de l'action correcte sur une variable au mauvais moment ou la réalisation d'une action incorrecte sur les données dans une variable. Ces anomalies comprennent :

- Ne pas réussir à attribuer une valeur à une variable avant de l'utiliser
- Prendre un chemin incorrect en raison d'une valeur incorrecte dans un prédicat de contrôle
- Essayer d'utiliser une variable après sa destruction
- Référencer une variable lorsqu'elle est hors de portée
- Déclarer et détruire une variable sans l'utiliser
- Redéfinir une variable avant qu'elle n'ait été utilisée
- Ne pas tuer une variable dynamiquement allouée (causant une possible fuite de mémoire)
- Modifier une variable, ce qui entraîne des effets secondaires inattendus (par exemple, les effets d'entraînement lors de la modification d'une variable globale sans tenir compte de toutes les utilisations de la variable)

Le langage de programmation utilisé peut guider les règles utilisées dans l'analyse du flot de données. Les langages de programmation peuvent permettre au programmeur d'effectuer sur les variables certaines actions qui ne sont pas illégales, mais qui peuvent amener le système à avoir un comportement différent de celui prévu par le développeur dans certaines circonstances. Par exemple, une variable peut être définie deux fois sans être réellement utilisée lorsqu'un certain chemin est suivi. L'analyse du flot de données étiquettera souvent ces utilisations comme « suspectes ». Bien qu'il puisse s'agir d'une affectation autorisée pour la variable, cela peut causer des problèmes de maintenabilité dans le code.

Le test du flot de données « utilise le graphe de flot de contrôle pour explorer les choses déraisonnables qui peuvent arriver aux données » [Beizer90] et trouve donc des défauts différents de l'analyse du flot de contrôle. Un Analyste Technique de Test devrait prévoir cette technique lors de la planification des tests puisque bon nombre de ces défauts causent des défaillances intermittentes difficiles à trouver lors de l'exécution de tests dynamiques.

L'analyse du flot de données est une technique statique ; elle ne permettra pas de mettre en évidence certains problèmes se produisant lorsque les données sont utilisées à l'exécution du système. Par exemple, la variable de donnée statique peut contenir un pointeur dans un tableau créé dynamiquement qui n'existe même pas avant l'exécution. L'utilisation de plusieurs processeurs et la préemption des tâches peuvent créer des conditions d'exécution problématiques qui ne seront pas détectées par l'analyse du flot de contrôle et du flot de données..

3.2.3 Utilisation de l'analyse statique pour améliorer la maintenabilité

L'analyse statique peut être appliquée de plusieurs façons pour améliorer la maniabilité du code, de l'architecture et des sites Web.

Du code mal écrit, non commenté et non structuré a tendance à être plus difficile à maintenir. Il peut nécessiter plus d'efforts des développeurs pour localiser et d'analyser les défauts dans le code, et la modification du code pour corriger un défaut ou ajouter une nouvelle fonctionnalité peut entraîner d'autres défauts.

L'analyse statique est utilisée avec le support de l'outil pour améliorer la maintenabilité en vérifiant la conformité aux normes et recommandations de codage. Ces normes et lignes directrices décrivent les pratiques de codage requises telles que les conventions de nommage, les commentaires, l'indentation et la modularisation du code. Notez que les outils d'analyse statique soulèvent généralement des avertissements plutôt que de détecter des défauts. Ces avertissements peuvent émis même pour du code syntactiquement correct.

Des outils d'analyse statique peuvent être appliqués au code utilisé pour implémenter des sites Web afin de vérifier l'exposition possible à des vulnérabilités de sécurité telles que l'injection de code, la sécurité des cookies, les scripts inter-sites, la falsification des ressources et l'injection de code SQL. De plus amples détails sont fournis à la section 4.3 et dans le syllabus de tests de sécurité de niveau avancé [ISTQB_ ALSEC_SYL].

Les conceptions modulaires entraînent généralement un code plus maintenable. Les outils d'analyse statique facilitent le développement du code modulaire de la manière suivante :

- Ils recherchent du code répété. Ces sections de code peuvent être des candidats à la refactorisation en modules (bien que le dépassement à l'exécution imposé par les appels de modules puisse être un problème pour les systems temps réel).
- Ils génèrent des mesures qui sont de précieux indicateurs de modularisation du code. Il s'agit notamment de mesures de couplage et de cohésion. Un système qui doit avoir une bonne maintenabilité aura probablement une faible mesure de couplage (le degré selon lequel les modules s'appuient les uns sur les autres pendant l'exécution) et une grande cohésion (le degré selon lequel un module est autonome et axé sur une seule tâche).
- Ils indiquent, dans le code orienté objet, où les objets dérivés peuvent avoir trop ou trop peu de visibilité dans les classes parentes.
- Ils mettent en évidence des zones de code ou d'architecture avec un haut niveau de complexité structurelle.

Le maintenance d'un site Web peut également être prise en charge à l'aide d'outils d'analyse statique. Ici, l'objectif est de vérifier si la structure arborescente du site est bien équilibrée ou s'il y a un déséquilibre qui sera la cause de :

- Tâches de test plus difficiles
- Augmentation de la charge de travail de maintenance
- Navigation difficile pour l'utilisateur

3.2.4 Graphes d'appel

Les graphes d'appel sont une représentation statique de la complexité de la communication. Il s'agit de graphes dirigés dans lesquels les nœuds représentent des modules de programme et les arcs représentent la communication entre ces modules.

Les graphes d'appel peuvent être utilisés dans les tests unitaires où différentes fonctions ou méthodes s'appellent les unes les autres, dans l'intégration et les tests système lorsque des modules distincts s'appellent les uns les autres, ou dans les tests d'intégration du système lorsque des systèmes distincts s'appellent les uns les autres.

Les graphiques d'appel peuvent être utilisés aux fins suivantes :

- Concevoir des tests qui appellent un module ou un système spécifique

- Établir le nombre d'emplacements dans le logiciel à partir desquels un module ou un système est appelé
- Évaluer la structure du code et l'architecture du système
- Fournir des suggestions pour l'ordre d'intégration (p.ex., intégration par paires et intégration par voisinage, comme nous l'avons vu ci-dessous)

Dans le syllabus du niveau Fondation [ISTQB_FL_SYL], deux catégories différentes de tests d'intégration ont été discutées : incrémentale (de haut en bas, de bas en haut.) et non incrémentale (big bang). Il est dit que les méthodes incrémentales sont préférables parce qu'elles introduisent du code par incréments, ce qui facilite l'isolement des défauts puisque la quantité de code en cause est limitée.

Dans ce syllabus avancé, trois autres méthodes non incrémentales utilisant des graphes d'appel sont introduites. Elles peuvent être préférables à des méthodes incrémentales qui nécessiteront probablement des builds supplémentaires pour terminer les tests et l'écriture de code non livrables pour les tests. Ces trois méthodes sont :

- Le Test d'intégration par paires (à ne pas confondre avec la technique de test de boîte noire « test pairwise »), qui cible les paires de composants qui fonctionnent ensemble comme on le voit dans le graphe d'appel pour les tests d'intégration. Bien que cette méthode ne réduise le nombre de builds que d'une petite quantité, elle réduit la quantité de code de harnais de test nécessaire.
- Les tests d'intégration par voisinage qui testent tous les nœuds qui se connectent à un nœud donné comme base pour les tests d'intégration. Tous les nœuds prédécesseurs et successeurs d'un nœud spécifique dans le graphe d'appel servent de base au test.
- L'approche fondée sur le prédicat conception de McCabe qui utilise la théorie de la complexité cyclomatique appliquée à un graphe d'appel pour modules. Cela nécessite la construction d'un graphe d'appel qui montre les différentes façons dont les modules peuvent s'appeler les uns les autres, et notamment :
    - Appel inconditionnel : l'appel d'un module à un autre se produit toujours
    - Appel conditionnel : l'appel d'un module à un autre arrive parfois
    - Appel conditionnel mutuellement exclusif : un module appellera un (et un seul) module parmi un certain nombre de modules différents
    - Appel itératif : un module en appelle un autre au moins une fois mais peut l'appeler plusieurs fois
    - Appel conditionnel itératif : un module peut en appeler un autre zéro ou plusieurs fois

  Après la création du graphe d'appel, la complexité d'intégration est calculée et des tests sont créés pour couvrir le graphe.

Se référer à [Jorgensen07] pour plus d'informations sur l'utilisation des graphes d'appel et des tests d'intégration par voisinage.

3.3 Analyse dynamique

3.3.1 Aperçu

L'analyse dynamique est utilisée pour détecter les défaillances dont les symptômes ne sont visibles que lorsque le code est exécuté. Par exemple, la possibilité de fuites mémoire peut être détectable par analyse statique (trouver du code qui alloue mais ne libère jamais de mémoire), mais une fuite mémoire est facilement mise en évidence par l'analyse dynamique.

Les défaillances qui ne sont pas immédiatement reproductibles (intermittentes) peuvent avoir des conséquences importantes sur l'effort de test et sur la capacité à livrer ou à utiliser des logiciels de manière productive. Ces défaillances peuvent être causées par des fuites de mémoire ou de ressources, une utilisation incorrecte de pointeurs et d'autres corruptions (p. ex., de la pile du système) [Kaner02].

En raison de la nature de ces défaillances, qui peuvent inclure l'aggravation progressive des performances du système ou même des plantages système, les stratégies de test doivent tenir compte des risques associés à ces défauts et, le cas échéant, effectuer une analyse dynamique pour les réduire (généralement en utilisant des outils). Étant donné que ces défaillances sont souvent les plus coûteuses à trouver et à corriger, il est recommandé de commencer l'analyse dynamique au début du projet .

L'analyse dynamique peut être appliquée pour accomplir ce qui suit :

- Prévenir les défaillances en détectant des fuites de mémoire (voir la section 3.3.2) et des pointeurs sauvages (voir la section 3.3.3)
- Analyser les défaillances du système qui ne peuvent pas être facilement reproduites
- Évaluer le comportement du réseau
- Améliorer les performances du système en fournissant des informations sur le comportement du système à l'exécution qui peuvent être utilisées pour apporter des modifications éclairées

L'analyse dynamique peut être effectuée à n'importe quel niveau de test et nécessite des compétences techniques et système pour faire ce qui suit :

- Spécifier les objectifs de test de l'analyse dynamique
- Déterminer le bon moment pour commencer et arrêter l'analyse
- Analyser les résultats

Lors des tests système, les outils d'analyse dynamique peuvent être utilisés même si les Analystes Techniques de Test ont des compétences techniques minimales ; les outils utilisés créent généralement des logs complets qui peuvent être analysés par ceux qui ont les compétences techniques nécessaires.

3.3.2 Détection des fuites de mémoire

Une fuite de mémoire se produit lorsque les zones de mémoire (RAM) disponibles à un programme sont allouées par ce programme, mais ne sont pas libérées par la suite lorsqu'elles ne sont plus nécessaires. Cette zone de mémoire est laissée comme allouée et n'est pas disponible pour être réutilisée. Lorsque cela se produit fréquemment ou dans des situations de mémoire faible, le programme peut manquer de mémoire utilisable. Historiquement, la manipulation de la mémoire était de la responsabilité du programmeur. Toutes les zones de mémoire allouées dynamiquement devaient être correctement communiquées par le programme d'allocation pour éviter une fuite de mémoire. De nombreux environnements de programmation modernes incluent un « ramasse miettes » automatique ou semi-automatique où la mémoire allouée est libérée après utilisation sans l'intervention directe du programmeur. Isoler des fuites de mémoire peut être très difficile dans les cas où la mémoire allouée est libérée par le « ramasse miettes » automatique.

Les fuites mémoire causent des problèmes qui se développent au fil du temps et peuvent ne pas être immédiatement évidents. Cela peut être le cas si, par exemple, le logiciel a été récemment installé ou le système redémarré, ce qui se produit souvent pendant les tests. Pour ces raisons, les effets négatifs des fuites de mémoire peuvent n'être remarqués que lorsque le programme est en production.

Le symptôme principal d'une fuite mémoire est une aggravation constante du temps de réponse du système pouvant finalement entraîner une défaillance du système. Bien que ces défaillances peuvent être résolues par le redémarrage du système, cela n'est pas toujours pratique, voire impossible.

De nombreux outils d'analyse dynamique identifient les zones du code où des fuites de mémoire se produisent afin qu'elles puissent être corrigées. De simples moniteurs de mémoire peuvent également être utilisés pour obtenir un aperçu de la diminution de la mémoire disponible au fil du temps, bien qu'une analyse complémentaire soit toujours nécessaire pour déterminer la cause exacte du problème.

Il y a d'autres types de fuites qui devraient également être considérées. Par exemple, la gestion de fichiers, les sémaphores et les pools de connexion pour les ressources.

3.3.3 Détection des pointeurs sauvages

Les pointeurs « sauvages » d'un programme sont des pointeurs qui ne sont plus exacts et qui ne doivent pas être utilisés. Par exemple, un pointeur sauvage peut avoir « perdu » l'objet ou la fonction pointée ou ne pas indiquer la zone de mémoire prévue (p. ex., il indique une zone qui dépasse les limites allouées d'un tableau). Lorsqu'un programme utilise des pointeurs sauvages, diverses conséquences peuvent se produire, notamment :

- Le programme peut fonctionner comme prévu. Cela peut être le cas lorsque le pointeur sauvage accède à la mémoire qui n'est actuellement pas utilisée par le programme et est théoriquement « libre » et / ou contient une valeur raisonnable.
- Le programme peut planter. Dans ce cas, le pointeur sauvage peut avoir causé l'utilisation incorrecte d'une partie de la mémoire, essentielle à l'exécution du programme (par exemple, le système d'exploitation).
- Le programme ne fonctionne pas correctement parce que les objets requis par le programme ne peuvent pas être consultés. Dans ces conditions, le programme peut continuer à fonctionner, bien qu'un message d'erreur puisse être émis.
- Les données dans l'emplacement de mémoire peuvent être corrompues par le pointeur et des valeurs incorrectes utilisées par la suite (cela peut également représenter une menace pour la sécurité).

Notez que toute modification apportée à l'utilisation de la mémoire du programme (p. ex., un nouveau build à la suite d'un changement de logiciel) peut déclencher l'une ou l'autre des quatre conséquences énumérées ci-dessus. Ceci est particulièrement critique lorsque, initialement, le programme fonctionne comme prévu malgré l'utilisation de pointeurs sauvages, puis plante de façon inattendue (peut-être même en production) à la suite d'un changement de logiciel. Il est important de noter que de telles défaillances sont souvent les symptômes d'un défaut sous-jacent (c.-à-d. le pointeur sauvage) (Reportez-vous à [Kaner02], « Leçon 74 »). Les outils peuvent aider à identifier les pointeurs sauvages lorsqu''ils sont utilisés par le programme, indépendamment de leur impact sur l'exécution du programme. Certains systèmes operating ont des fonctions intégrées pour vérifier les violations de l'accès à la mémoire pendant l'exécution. Par exemple, le système d'exploitation peut émettre une exception lorsqu'une application tente d'accéder à un emplacement de mémoire qui se trouve en dehors de la zone de mémoire autorisée de cette application.

3.3.4 Analyse de l'efficacité de la performance

L'analyse dynamique n'est pas utile seulement pour détecter les défaillances. Grâce à l'analyse dynamique des performances du programme, les outils aident à identifier les goulets d'étranglement pour l'efficacité des performances et génèrent un large éventail de mesures de performance pouvant être utilisées par le développeur pour régler les performances du système. Par exemple, des informations peuvent être fournies sur le nombre de fois qu'un module est appelé pendant l'exécution. Les modules fréquemment appelés seraient probablement des candidats à l'amélioration du rendement.

En combinant les informations sur le comportement dynamique du logiciel avec les informations obtenues à partir de graphes d'appel lors de l'analyse statique (voir Section 3. 2.4), le testeur peut également identifier les modules qui pourraient être candidats à des tests plus détaillés et approfondis (par exemple, les modules qui sont fréquemment appelés et ont de nombreuses interfaces).

L'analyse dynamique de la performance du programme se fait souvent lors des tests système, bien qu'elle puisse également être faite lors du test d'un sous-système unique dans les phases antérieures de test à l'aide d'un harnais de test. De plus amples détails sont fournis dans le syllabus tests de performance au niveau Fondation [ISTQB_FLPT_SYL].

4. Caractéristiques Qualité pour les Tests Techniques - 345 mins.

Mots clés

responsabilité, adaptabilité, facilité d'analyse, authenticité, disponibilité, capacité, co-existence, compatibilité, confidentialité, tolérance aux fautes, facilité d'installation, intégrité, maintenabilité, maturité, facilité de modification, modularité, non-répudiation, test d'acceptation opérationnelle, profil opérationnel, efficacité de la performance, portabilité, caractéristique qualité, récupération, fiabilité, model de croissance de fiabilité, facilité de remplacement, utilisation des ressources, réutilisabilité, sécurité, testabilité, comportement dans le temps

Objectifs d'apprentissage pour Caractéristiques Qualité pour les Tests Techniques

4.2 Questions générales de planification

TTA-4.2.1 (K4) Pour un scénario particulier, analyser les exigences non-fonctionnelles et rédiger les sections respectives du plan de test
TTA-4.2.2 (K3) Compte tenu d'un risque produit particulier, définir les types de test non fonctionnel particulier qui sont les plus appropriés
TTA-4.2.3 (K2) Comprendre et expliquer les étapes du cycle de vie du développement logiciel d'une application où les tests non fonctionnels doivent généralement être appliqués
TTA-4.2.4 (K3) Pour un scénario donné, définir les types de défauts que vous vous attendez à trouver en utilisant les différents types de tests non fonctionnels

4.3 Tests de sécurité

TTA-4.3.1 (K2) Expliquer les raisons d'inclure des tests de sécurité dans une approche de test
TTA-4.3.2 (K2) Expliquer les principaux aspects à prendre en compte dans la planification et la spécification des tests de sécurité

4.4 Tests de fiabilité

TTA-4.4.1 (K2) Expliquer les raisons d'inclure des tests de fiabilité dans une approche de test
TTA-4.4.2 (K2) Expliquer les principaux aspects à prendre en compte dans la planification et la spécification des tests de fiabilité

4.5 Tests de performance

TTA-4.5.1 (K2) Expliquer les raisons d'inclure des tests de performance dans une approche de test
TTA-4.5.2 (K2) Expliquer les principaux aspects à prendre en compte dans la planification et la spécification des tests de performance

4.6 Tests de maintenabilité

TTA-4.6.1 (K2) Expliquer les raisons d'inclure des tests de maintenabilité dans une approche de test

4.7 Test de portabilité

TTA-4.7.1 (K2) Expliquer les raisons d'inclure des tests de portabilité dans une approche de test

4.8 Test de compatibilité

TTA-4.8.1 (K2) Expliquer les raisons d'inclure des tests de compatibilité dans une approche de test

4.1 Introduction

En général, l'Analyste Technique de Test concentre les tests sur "comment" le produit fonctionne, plutôt que les aspects fonctionnels du « quoi ». Ces tests peuvent avoir lieu à n'importe quel niveau de test. Par exemple, lors du test de composants de systèmes temps réel et embarqués, il est important d'effectuer des analyses comparatives de l'efficacité des performances et de tester l'utilisation des ressources. Lors des tests d'acceptation opérationnelle, il convient de tester les aspects de fiabilité, tels que la récupérabilité. Les tests à ce niveau visent à tester un système spécifique, c'est-à-dire des combinaisons de matériel et de logiciels. Le système spécifique sous test peut inclure divers serveurs, clients, bases de données, réseaux et autres ressources. Quel que soit le niveau de test, les tests doivent être effectués en fonction des priorités de risque et des ressources disponibles.

Il convient de noter que des tests dynamiques et statiques (voir le chapitre 3) peuvent être appliqués pour tester les caractéristiques de qualité non fonctionnelles décrites dans le présent chapitre.

La description des caractéristiques qualité des produits fournie dans ISO 25010 [ISO25010] est utilisée comme guide pour décrire les caractéristiques et leurs sous-caractéristiques. Celles-ci sont indiquées dans le tableau ci-dessous, ainsi qu'une indication des caractéristiques/sous-caractéristiques couvertes par les syllabus Analyste de Test et Analyste Technique de Test.

Caractéristiques	Sous-caractéristiques	Analyste de Test	Analyste Technique de Test
Aptitude fonctionnelle	Exactitude fonctionnelle, adéquation fonctionnelle, complétude fonctionnelle	X	
Fiabilité	Maturité, tolérance aux défauts, récupération, disponibilité		X
Utilisabilité	Reconnaissance de la pertinence, apprentissage, opérabilité, esthétique de l'interface utilisateur, protection contre les erreurs de l'utilisateur, accessibilité	X	
Efficacité de la performance	Comportement dans le temps, utilisation des ressources, capacité		X
Maintenabilité	Facilité d'analyse, facilité de modification, testabilité, modularité, réutilisabilité		X
Portabilité	Adaptabilité, facilité d'installation, facilité de remplacement	X	X
Sécurité	Confidentialité, intégrité, non-répudiation, responsabilité, authenticité		X
Compatibilité	Coexistence		X
	Interopérabilité	X	

Notez qu'un tableau est fourni à l'Annexe A qui compare les caractéristiques décrites dans l'ISO 9126 (tel qu'utilisé dans la version 2012 de ce programme) avec ceux du nouvel ISO 25010.

Pour toutes les caractéristiques et les sous-caractéristiques de qualité abordées dans cette section, les risques typiques doivent être reconnus afin qu'une stratégie de test appropriée puisse être formée et documentée. Les tests des caractéristiques de qualité nécessitent une attention particulière au cycle de vie de développement logiciel, au calendrier, aux outils requis, à la disponibilité des logiciels et de la documentation, et à l'expertise technique. En l'absence d'une stratégie pour répondre à chaque caractéristique et à ses besoins uniques en matière de tests, le testeur peut ne pas avoir suffisamment de temps dans son planning pour la planification, la montée en compétence et le temps d'exécution des tests [Bath14].

Certains de ces tests, par exemple, les tests de performance, nécessitent une planification, des équipements dédiés, des outils spécifiques, des compétences spécialisées en test et, dans la plupart des cas, beaucoup de temps. Le test des caractéristiques et des sous-caractéristiques qualité doit être intégré dans le calendrier global des tests avec des ressources adéquates allouées à l'effort. Chacun de ces types de test a des besoins spécifiques, cible des problèmes spécifiques et peut se produire à différents moments du cycle de vie du développement logiciel, comme discuté dans les sections ci-dessous.

Alors que le Test Manager s'intéresse à la compilation et à la production de rapports synthétiques présentant les mesures de métriques concernant les caractéristiques et les sous-caractéristiques qualité, l'Analyste de Test ou l'Analyste Technique de Test (selon le tableau ci-dessus) recueille l'information pour chaque mesure.

Les mesures des caractéristiques de qualité recueillies lors des tests de pré-production par l'Analyste Technique de Test peuvent servir de base aux « SLA » (Accord de Niveau de Service) entre le fournisseur et les parties prenantes (p. ex., clients, opérateurs) du système logiciel. Dans certains cas, les tests peuvent continuer à être exécutés après l'entrée en production du logiciel, souvent par une équipe ou une organisation distincte. Ceci se fait généralement pour l'efficacité de la performance et les tests de fiabilité qui peuvent montrer des résultats différents dans l'environnement de production que dans l'environnement de test.

4.2 Questions générales de planification

L'incapacité de planifier des tests non fonctionnels peut mettre le succès d'une application en grand danger. Le Test Manager peut demander à l'Analyste Technique de Test d'identifier les principaux risques pour les caractéristiques de qualité pertinentes (voir tableau dans la section 4.1) et d'aborder tous les problèmes de planification associés aux tests proposés. Cette information peut être utilisée dans la création du plan de test maître.

Les facteurs généraux suivants sont pris en compte lors de l'exécution de ces tâches :

- Exigences des parties prenantes
- Acquisition des outils nécessaires et formations associées
- Exigences en matière d'environnement de test
- Considérations organisationnelles
- Considérations liées à la sécurité des données
- Risques et défauts typiques

4.2.1 Exigences des parties prenantes

Les exigences non fonctionnelles sont souvent mal spécifiées, voire inexistantes. À l'étape de la planification, les Analystes Techniques de Test doivent être en mesure d'obtenir des niveaux d'attente relatifs aux caractéristiques techniques de qualité des parties prenantes concernées et d'évaluer les risques associés.

Une approche commune consiste à supposer que si le client est satisfait de la version existante du système, il continuera d'être satisfait des nouvelles versions, tant que les niveaux de qualité atteints seront maintenus. Cela permet d'utiliser la version existante du système comme point de référence. Il peut s'agir d'une approche particulièrement utile à adopter pour certaines des caractéristiques de qualité non fonctionnelles telles que l'efficacité de la performance, où les intervenants peuvent avoir de la difficulté à préciser leurs exigences.

Il est conseillé d'obtenir plusieurs points de vue lors de la capture des exigences non fonctionnelles. Elles doivent être obtenues auprès d'intervenants tels que les clients, les Product Owners, les utilisateurs, le personnel d'exploitation et le personnel de maintenance. Si les principaux intervenants

sont manquants, certaines exigences risquent d'être manquées. Pour plus de détails sur les exigences de capture, consultez le Syllabus Avancé Test Manager [ISTQB_ALTM_SYL].

Dans les projets en Agile, les exigences non fonctionnelles peuvent être indiquées comme des User Stories ou ajoutées aux fonctionnalité spécifiées dans les cas d'utilisation comme contraintes non fonctionnelles.

4.2.2 Acquisition des outils nécessaires et formations associées

Les outils commerciaux ou les simulateurs sont particulièrement pertinents pour les tests d'efficacité des performances et certains tests de sécurité. Les Analystes Techniques de Test devraient estimer les coûts et les délais associés à l'acquisition, à l'apprentissage et à la mise en œuvre des outils. Lorsque des outils spécialisés doivent être utilisés, la planification devrait tenir compte des courbes d'apprentissage des nouveaux outils et/ou du coût de l'embauche de spécialistes externes des outils.

Le développement d'un simulateur complexe peut représenter un projet de développement à part entière et devrait être planifié en tant que tel. En particulier, les tests et la documentation de l'outil développé doivent être pris en compte dans le calendrier et l'affectation des ressources. Un budget et un temps suffisants devraient être prévus pour la mise à niveau et le retest du simulateur au fur et à mesure que le produit simulé change. La planification des simulateurs à utiliser dans des applications critiques pour la sécurité doit tenir compte des tests d'acceptation et de la certification possible du simulateur par un organisme indépendant.

4.2.3 Exigences en matière d'environnement de test

De nombreux tests techniques (p. ex., tests de sécurité, tests d'efficacité des performances) nécessitent un environnement de test de qualité représentatif de la production afin de fournir des mesures réalistes. Selon la taille et la complexité du système sous test, cela peut avoir une incidence importante sur la planification et le financement des tests. Étant donné que le coût de ces environnements peut être élevé, les solutions alternatives suivantes peuvent être considérées :

- Utilisation de l'environnement de production
- Utilisation d'une version réduite du système, en prenant soin que les résultats des tests obtenus soient suffisamment représentatifs du système de production
- Utilisation de ressources basées sur le cloud comme alternative à l'acquisition directe des ressources
- Utilisation d'environnements virtualisés

La programmation de ces exécutions de test doit être planifiée avec soin et il est fort probable que ces tests ne puissent être exécutés qu'à des moments spécifiques (par exemple, à de faibles heures d'utilisation).

4.2.4 Considérations organisationnelles

Les tests techniques peuvent consister à mesurer le comportement de plusieurs composants d'un système complet (p. ex., serveurs, bases de données, réseaux). Si ces composantes sont réparties sur un certain nombre de sites et d'organisations différents, les efforts nécessaires pour planifier et coordonner les tests peuvent être importants. Par exemple, certains composants logiciels peuvent n'être disponibles pour les tests système qu'à des moments particuliers de la journée ou de l'année, ou les organisations ne peuvent offrir une prise en charge pour les tests que pour un nombre limité de jours. Ne pas réussir à garantir la disponibilité « sur demande » de composantes du système ou de personnel (c.-à-d. d'expertise « empruntée ») d'autres organisations à des fins de test peut entraîner de graves perturbations des tests prévus.

4.2.5 Considérations relatives à la sécurité des données

Les mesures de sécurité spécifiques mises en œuvre pour un système devraient être prises en compte à l'étape de la planification des tests afin de s'assurer que toutes les activités de test sont possibles. Par exemple, l'utilisation du chiffrement des données peut rendre difficile la création de données de test et la vérification des résultats.

Les politiques et les réglements en matière de protection des données peuvent empêcher la génération de données de test sur la base des données de production (p. ex., données personnelles, données de carte de crédit). Rendre les données de test anonymes est une tâche non triviale qui doit être planifiée dans le cadre de l'implémentation des tests.

4.2.6 Risques et défauts typiques

L'identification et la gestion des risques sont une considération fondamentale pour la planification des tests (voir le chapitre 1). L'Analyste Technique de Test identifie les risques produit en utilisant la connaissance des types typiques de défauts à prévoir pour une caractéristique de qualité particulière. Cela permet de sélectionner les types de tests nécessaires pour traiter ces risques. Ces aspects spécifiques sont abordés dans les sections restantes de ce chapitre qui décrivent chacune des caractéristiques de qualité.

4.3 Tests de sécurité

4.3.1 Raisons d'envisager des tests de sécurité

Les tests de sécurité évaluent la vulnérabilité d'un système aux menaces en essayant de compromettre la politique de sécurité du système. On peut citer comme menaces potentielles qui devraient être explorées lors des tests de sécurité :

- Copie non autorisée d'applications ou de données.
- Contrôle d'accès non autorisé (p. ex., capacité d'effectuer des tâches pour lesquelles l'utilisateur n'a pas de droits). Les droits, l'accès et les privilèges des utilisateurs sont au centre de ces tests. Ces informations devraient être disponibles dans les spécifications du système.
- Logiciel qui présente des effets secondaires involontaires lors de l'exécution de sa fonction prévue. Par exemple, un lecteur multimédia qui lit correctement l'audio, mais le fait en écrivant des fichiers sur le stockage temporaire non crypté présente un effet secondaire qui peut être exploité par des pirates logiciels.
- Code inséré dans une page Web et pouvant être exercé par les utilisateurs suivants (script cross-site ou XSS). Ce code peut être malveillant.
- Débordement tampon (dépassement tampon - buffer overflow) qui peut être causé par l'entrée de chaînes de caractères dans un champ d'entrée d'interface utilisateur qui sont plus longues que ce que le code peut gérer correctement. Une vulnérabilité de débordement tampon représente une opportunité pour l'exécution d'instructions de code malveillantes.
- Déni de service, qui empêche les utilisateurs d'interagir avec une application (par exemple, en surchargeant un serveur Web de demandes de « nuisance »).
- L'interception, l'imitation et/ou la modification et le relais subséquent de communications (p. ex., transactions par carte de crédit) par un tiers de sorte qu'un utilisateur ne soit pas au courant de la présence de ce tiers (attaque « Man-in-the-middle»)
- Briser les codes de cryptage utilisés pour protéger les données sensibles.
- Bombes logiques (parfois appelées œufs de Pâques), qui peuvent être insérées malicieusement dans le code et qui ne s'activent que sous certaines conditions (par exemple, à une date spécifique). Lorsque des bombes logiques s'activent, elles peuvent effectuer des actes malveillants tels que la suppression de fichiers ou le formattage de disques.

4.3.2 Planification des tests de sécurité

En général, les aspects suivants sont particulièrement pertinents lors de la planification des tests de sécurité :

- Étant donné que des problèmes de sécurité peuvent être introduits au cours de l'architecture, de la conception et de l'implémentation du système, des tests de sécurité peuvent être programmés pour les niveaux de test composant, intégration et système. En raison de la nature changeante des menaces de sécurité, des tests de sécurité peuvent également être programmés régulièrement après l'entrée en production du système. Cela est particulièrement vrai pour les architectures ouvertes dynamiques telles que l'Internet des objets (IoT) où la phase de production se caractérise par de nombreuses mises à jour des éléments logiciels et matériels utilisés.
- Les approches de test proposées par l'Analyste Technique de Test peuvent inclure des revues de l'architecture, de la conception et du code, et l'analyse statique du code avec des outils de sécurité. Celles-ci peuvent être efficaces pour trouver des problèmes de sécurité qui sont facilement manqués lors des tests dynamiques.
- L'Analyste Technique de Test peut être appelé à concevoir et à effectuer certaines « attaques » de sécurité (voir ci-dessous) qui nécessitent une planification et une coordination minutieuses avec les parties prenantes (y compris les spécialistes des tests de sécurité). D'autres tests de sécurité peuvent être effectués en coopération avec les développeurs ou avec les Analystes de Test (par exemple, tester les droits des utilisateurs, l'accès et les privilèges).
- Un aspect essentiel de la planification des tests de sécurité est l'obtention d'approbations. Pour l'Analyste Technique de Test, cela signifie s'assurer que l'autorisation explicite d'effectuer les tests de sécurité prévus a été obtenue par le Test Manager. Tout autre test non planifié effectué pourrait être considéré comme des attaques réelles et la personne qui effectue ces tests pourrait risquer une action en justice. Sans écrit pour montrer l'intention et l'autorisation, l'excuse « Nous avons effectué un test de sécurité » a peu de chance d'être convaincante.
- Toute planification de tests de sécurité doit être coordonnée avec le responsable de sécurité de l'information d'une organisation si dans l'organisation un tel rôle existe.
- Il convient de noter que les améliorations pouvant être apportées à la sécurité d'un système peuvent affecter efficacité de ses performances ou sa fiabilité. Après avoir apporté des améliorations de sécurité, il est conseillé de tenir compte de la nécessité d'effectuer des tests d'efficacité des performances ou de fiabilité (voir Sections 4.4 et 4. 5 ci-dessous).

Des normes spécifiques peuvent s'appliquer lors de la planification des tests de sécurité, tels que [ISA/IEC 62443-3-2] qui s'applique aux systèmes industriels d'automatisation et de contrôle.

Le Syllabus de tests de sécurité de niveau avancé [ISTQB_ALSEC_SYL] comprend plus de détails sur les éléments clé d'un plan de test de sécurité.

4.3.3 Spécification des tests de sécurité

Des tests de sécurité particuliers peuvent être regroupés [Whittaker04] en fonction de l'origine du risque pour la sécurité. Il s'agit notamment des éléments suivants :

- Interface Utilisateur : Accès non autorisé et entrées malveillantes
- Fichiers système : accès aux données sensibles stockées dans des fichiers ou des référentiels
- Système d'exploitation : stockage d'informations sensibles telles que des mots de passe sous forme non cryptée en mémoire qui pourraient être exposés lorsque le système est écrasé par des entrées malveillantes
- Logiciels externes : interactions pouvant se produire entre les composants externes que le système utilise. Ceux-ci peuvent se trouver au niveau du réseau (p. ex., paquets ou messages incorrects transmis) ou au niveau des composants logiciels (p. ex., défaillance d'un composant logiciel sur lequel le logiciel s'appuie)

Les sous-caractéristiques de sécurité de l'ISO 25010 [ISO25010] fournissent également une base à partir de laquelle les tests de sécurité peuvent être spécifiés. Ceux-ci se concentrent sur les aspects suivants de la sécurité :

- Confidentialié – la mesure selon laquelle un produit ou un système garantit que les données ne sont accessibles qu'aux personnes autorisées à avoir accès
- Intégrité – mesure selon laquelle un système, un produit ou un composant empêche l'accès non autorisé ou la modification de programmes ou de données informatiques
- Non-répudiation – la mesure selon laquelle il peut être prouvé que des actions ou des événements ont eu lieu de sorte qu'ils ne peuvent pas être niés plus tard
- Responsabilité – la mesure selon laquelle les actions d'une entité peuvent être tracées individuellement à l'entité
- Authenticité – la mesure selon laquelle l'identité d'un sujet ou d'une ressource peut s'avérer être celle revendiquée

L'approche suivante [Whittaker04] peut être utilisée pour développer des tests de sécurité :

- Recueillir des informations qui peuvent être utiles pour spécifier des tests, tels que les noms des employés, les adresses physiques, les détails concernant les réseaux internes, les numéros IP, l'identité du logiciel ou du matériel utilisé, et la version du système d'exploitation.
- Effectuez une analyse de vulnérabilité à l'aide d'un des nombreux outils disponibles. Ces outils ne sont pas utilisés directement pour compromettre le système, mais pour identifier les vulnérabilités qui sont, ou qui peuvent entraîner, une violation de la politique de sécurité. Des vulnérabilités spécifiques peuvent également être identifiées à l'aide d'informations et de checklists telles que celles fournies par le National Institute of Standards and Technology (NIST) [Web-1] et le Open Web Application Security Project™ (OWASP) [Web-4].
- Élaborer des « plans d'attaque » (c.-à-d. un plan d'actions de test visant à compromettre la politique de sécurité d'un système particulier) à l'aide des informations recueillies. Plusieurs entrées via différentes interfaces (par exemple, interface utilisateur, système de fichiers) doivent être spécifiées dans les plans d'attaque pour détecter les défauts de sécurité les plus graves. Les diverses « attaques » décrites dans [Whittaker04] sont une source précieuse de techniques développées spécifiquement pour les tests de sécurité.

Notez que des plans d'attaque peuvent être élaborés pour les tests de pénétration (voir [ISTQB_ALSEC_SYL]).

Les problèmes de sécurité peuvent également être mis en évidence par des revues (voir le chapitre 5) et/ou l'utilisation d'outils d'analyse statique (voir Section 3.2). Les outils d'analyse statique contiennent un ensemble étendu de règles spécifiques aux menaces de sécurité et contre lesquelles le code est vérifié. Par exemple, les problèmes de débordement tampon, causés par une défaillance dans la vérification de la taille du tampon avant l'affectation des données, peuvent être trouvés par l'outil.

La section 3.2 (analyse statique) et le syllabus niveau avancé tests de sécurité [ISTQB_ALSEC_SYL] comprennent plus de détails sur les tests de sécurité.

4.4 Tests de fiabilité

4.4.1 Introduction

La classification ISO 25010 des caractéristiques de qualité du produit définit les sous-caractéristiques suivantes de fiabilité:

- Maturité – le degré auquel un composant ou un système répond aux besoins de fiabilité en fonctionnement normal
- Tolérance aux fautes - la capacité du produit logiciel à maintenir un niveau spécifié de performance en cas de défauts logiciels ou de violation de son interface spécifiée

- Récupération - la capacité du produit logiciel à rétablir un niveau spécifié de performance et à récupérer les données directement affectées en cas de défaillance
- Disponibilité – le degré auquel un composant ou un système est opérationnel et accessible lorsque cela est nécessaire pour une utilisation

4.4.2 Mesure de la maturité du logiciel

L'un des objectifs des tests de fiabilité est de surveiller une mesure statistique de maturité du logiciel au fil du temps et de la comparer à un objectif de fiabilité qui peut être exprimé sous la forme d'un accord de niveau de service (SLA : Service Level Agreement). Les mesures peuvent prendre la forme d'un temps moyen entre les défaillances (MTBF), d'un temps moyen pour réparer (MTTR) ou de toute autre forme de mesure de l'intensité des défaillances (p. ex., nombre de défaillances d'une sévérité particulière qui se produisent par semaine). Celles-ci peuvent être utilisées comme critères de sortie (p. ex., pour la livraison en production).

Notez que la maturité dans le contexte de la fiabilité ne doit pas être confondue avec la maturité de l'ensemble du processus de test logiciel, comme abordé dans le syllabus ISTQB ® Niveau Avancé Test Manager [ISTQB_ALTM_SYL].

4.4.3 Test de tolérance aux fautes

En plus des tests fonctionnels qui évaluent la tolérance du logiciel aux fautes en termes de traitement de valeurs d'entrée inattendues (tests dits négatifs), des tests supplémentaires sont nécessaires pour évaluer la tolérance d'un système aux fautes qui se produisent à l'extérieur de l'application testée. Ces fautes sont généralement signalées par le système d'exploitation (p. ex., disque plein, processus ou service non disponible, fichier non trouvé, mémoire non disponible). Les tests de tolérance aux fautes au niveau du système peuvent être soutenus par des outils spécifiques.

Notez que les termes « robustesse » et « tolérance aux erreurs » sont également couramment utilisés pour discuter de la tolérance aux fautes (voir [ISTQB_GLOSSARY] pour plus de détails).

4.4.4 Test de récupération

D'autres formes de tests de fiabilité évaluent la capacité du système logiciel à se remettre de défaillances matérielles ou logicielles d'une manière prédéterminée, ce qui permet par la suite de reprendre les opérations normales. Les tests de récupération comprennent les tests de basculement après défaillance, de sauvegarde et de restauration.

Les tests de basculement après défaillance sont effectués lorsque les conséquences d'une défaillance logicielle sont si négatives que des mesures matérielles et/ou logicielles spécifiques ont été mises en œuvre pour assurer le fonctionnement du système même en cas de défaillance. Des tests de basculement peuvent s'appliquer, par exemple, lorsque le risque de pertes financières est extrême ou lorsqu'il existe des problèmes de sécurité critiques. Lorsque des défaillances peuvent provoquer des événements catastrophiques, cette forme de test de récupération peut également être appelée test de « récupération après sinistre ».

Les mesures préventives typiques pour les défaillances matérielles peuvent inclure la répartition de la charge (load balancing) entre plusieurs processeurs et le regroupement (clustering) de serveurs, processeurs ou disques afin que l'un puisse immédiatement prendre le relais d'un autre s'il échoue (systèmes redondants). Une mesure logicielle typique pourrait être la mise en œuvre de plus d'une instance indépendante d'un système logiciel (par exemple, le système de commande de vol d'un aéronef) dans des systèmes dissemblables dits redondants. Les systèmes redondants sont généralement une combinaison de mesures logicielles et matérielles et peuvent être appelés systèmes duplex, triplex ou quadruplex, selon le nombre d'instances indépendantes (deux, trois ou quatre

respectivement). L'aspect dissemblable pour le logiciel est atteint lorsque les mêmes exigences logicielles sont fournies à deux (ou plusieurs) équipes de développement indépendantes et non connectées, dans le but d'avoir les mêmes services fournis avec des logiciels différents. Cela protège les systèmes dissemblables redondants en ce qu'une entrée défectueuse similaire est moins susceptible d'avoir le même résultat. Ces mesures prises pour améliorer la capacité de récupération d'un système peuvent influer directement sur sa fiabilité et devraient également être prises en compte lors des tests de fiabilité.

Les tests de basculement sont conçus pour tester explicitement les systèmes en simulant des modes de défaillance ou en causant des défaillances dans un environnement contrôlé. À la suite d'une défaillance, le mécanisme de basculement est testé pour s'assurer que les données ne sont pas perdues ou corrompues et que tous les niveaux de service convenus sont maintenus (p. ex., disponibilité des fonctions ou temps de réponse).

Les tests de sauvegarde et restauration se concentrent sur les mesures procédurales mises en place pour minimiser les effets d'une défaillance. Ces tests évaluent les procédures (habituellement documentées dans un manuel) pour prendre différentes formes de sauvegarde et pour restaurer ces données en cas de perte ou de corruption de données. Les cas de test sont conçus pour s'assurer que les chemins critiques à travers chaque procédure sont couverts. Des revues techniques peuvent être effectuées pour « exécuter à blanc » ces scénarios et valider les manuels par rapport aux procédures réelles. Des tests d'acceptation opérationnelle exercent les scénarios dans un environnement de production ou reproductif de la production pour valider leur utilisation réelle.

Les mesures pour les tests de sauvegarde et de restauration peuvent inclure :
- Temps pris pour effectuer différents types de sauvegarde (p. ex., complète, incrémentale)
- Temps pris pour restaurer les données
- Niveaux de sauvegarde garantie des données (p. ex., récupération de toutes les données d'au plus 24 heures, récupération de données de transaction spécifiques d'une heure au plus)

4.4.5 Tests de disponibilité

Tout système qui a des interfaces avec d'autres systèmes et/ou processus (par exemple, pour recevoir des entrées) s'appuie sur la disponibilité de ces interfaces pour assurer l'opérabilité globale.

Les tests de disponibilité servent les principaux objectifs suivants :
- Déterminer si les composants et processus requis du système sont disponibles (sur demande ou en continu) et répondent comme prévu aux demandes
- Fournir des mesures à partir desquelles un niveau global de disponibilité peut être obtenu (souvent donné en pourcentage de temps dans un SLA).
- Déterminer si un système global est prêt à fonctionner (p. ex., comme l'un des critères pour les tests d'acceptation opérationnelle).

Les tests de disponibilité sont effectués avant et après la mise en service opérationnelle, et sont particulièrement pertinents pour les situations suivantes :
- Lorsque les systèmes sont composés d'autres systèmes (c.-à-d. systèmes de systèmes). Les tests se concentrent sur la disponibilité de tous les systèmes de composants individuels.
- Lorsqu'un système ou un service est fourni de façon externalisée (p. ex., par un fournisseur tiers). Les tests sont axés sur la mesure des niveaux de disponibilité afin de s'assurer que les niveaux de service convenus sont maintenus.

La disponibilité peut être mesurée à l'aide d'outils de surveillance dédiés ou en exécutant des tests spécifiques. Ces tests sont généralement automatisés et peuvent être exécutés parallèlement aux opérations normales à condition qu'ils n'aient pas d'incidence sur les opérations normales (p. ex., en réduisant l'efficacité de la performance).

4.4.6 Planification des tests de fiabilité

En général, les aspects suivants sont particulièrement pertinents lors de la planification des tests de fiabilité :

- La fiabilité peut continuer à être surveillée après la mise en production du logiciel. L'organisation et le personnel responsables de l'exploitation du logiciel doivent être consultés lors de la collecte des exigences de fiabilité à des fins de planification des tests.
- L'Analyste Technique de Test peut choisir un modèle d'amélioration de la fiabilité qui montre les niveaux de fiabilité attendus au fil du temps. Un modèle d'amélioration de la fiabilité peut fournir des informations utiles au Test Manager en permettant la comparaison des niveaux de fiabilité attendus et atteints.
- Les tests de fiabilité doivent être effectués dans un environnement représentatif de la production. L'environnement utilisé doit rester aussi stable que possible pour permettre de surveiller les tendances de fiabilité au fil du temps.
- Étant donné que les tests de fiabilité nécessitent souvent l'utilisation de l'ensemble du système, les tests de fiabilité sont le plus souvent effectués dans le cadre des tests système. Toutefois, les composants individuels peuvent être soumis à des tests de fiabilité de même que des ensembles intégrés de composants. Des examens détaillés de l'architecture, de la conception et du code peuvent également être utilisés pour éliminer certains risques de problèmes de fiabilité liés au système implémenté.
- Afin de produire des résultats de test qui sont statistiquement significatifs, les tests de fiabilité nécessitent généralement de longs temps d'exécution. Il peut donc être difficile de les planifier avec d'autres tests.

4.4.7 Spécification des tests de fiabilité

Les tests de fiabilité peuvent prendre la forme d'un ensemble répété de tests prédéterminés. Il peut s'agir de cas de tests sélectionnés au hasard à partir d'un référentiel ou de cas de test générés par un modèle statistique à l'aide de méthodes aléatoires ou pseudo-aléatoires. Les tests peuvent donc être basés sur des modèles d'utilisation que l'on appelle parfois « profils opérationnels »(voir section 4.5.3).

Lorsque des tests de fiabilité sont programmés pour s'exécuter automatiquement parallèlement aux opérations normales (p. ex., pour tester la disponibilité), ils sont généralement spécifiés pour être aussi simples que possible afin d'éviter un impact négatif possible sur l'efficacité des performances du système.

Certains tests de fiabilité peuvent spécifier que les actions utilisant beaucoup de mémoire soient exécutées à plusieurs reprises afin que d'éventuelles fuites mémoire puissent être détectées.

4.5 Tests de performance

4.5.1 Types de tests de performance

4.5.1.1 Test de charge

Les tests de charge mettent l'accent sur la capacité d'un système à gérer des niveaux croissants de charge réalistes prévus résultant des demandes de transaction générées par un nombre d'utilisateurs ou de processus simultanés. Les temps de réponse moyens des utilisateurs dans différents scénarios d'utilisation typique (profils opérationnels) peuvent être mesurés et analysés. Voir aussi [Splaine01].

4.5.1.2 Test de stress

Les tests de stress sont axés sur la capacité d'un système ou d'un composant à gérer des pics de charges à la limite ou au-delà de la charge de travail prévue ou spécifiée, ou sur la disponibilité réduite de ressources telles que la bande passante disponible. Les niveaux de performance devraient se

dégrader lentement et de façon prévisible sans défaillance à mesure que les niveaux de stress sont augmentés. En particulier, l'intégrité fonctionnelle du système doit être testée pendant que le système est stressé afin de trouver des défauts possibles dans le traitement fonctionnel ou les incohérences de données.

L'un des objectifs possibles des tests de stress est de découvrir les limites auxquelles un système échoue réellement afin que le « maillon le plus faible de la chaîne » puisse être déterminé. Les tests de stress permettent d'ajouter une capacité supplémentaire au système en temps opportun (p. ex., mémoire, capacité cpu, stockage de base de données).

4.5.1.3 Test d'évolutivité

Les tests d'évolutivité se concentrent sur la capacité d'un système à répondre à des besoins futurs en matière d'efficacité, qui peuvent être au-delà de ceux actuellement requis. L'objectif des tests est de déterminer la capacité du système à croître (p. ex., avec plus d'utilisateurs, de plus grandes quantités de données stockées) sans atteindre un point où les exigences de performances actuellement spécifiées ne peuvent pas être remplies ou où le système échoue. Une fois que les limites de l'évolutivité sont connues, des valeurs seuils peuvent être fixées et surveillées en production afin de fournir un avertissement en cas de problèmes imminents. En outre, l'environnement de production peut être ajusté avec la quantité appropriée de matériel pour répondre aux besoins prévus.

4.5.2 Planification des tests de performance

En plus des questions de planification générale décrites dans la section 4.2, les facteurs suivants peuvent influencer la planification des tests de performance :

- Selon l'environnement de test utilisé et le logiciel testé (voir section 4.2.3), les tests de performance peuvent nécessiter la mise en œuvre de l'ensemble du système avant que des tests efficaces puissent être effectués. Dans ce cas, des tests de performance sont généralement programmés pendant le test système. D'autres tests de performance peuvent être effectués efficacement au niveau des composants en étant programmés lors des tests unitaires.
- En général, il est souhaitable d'effectuer des tests initiaux d'efficacité de la performance dès que possible, même si un environnement reproductif de la production n'est pas encore disponible. Ces premiers tests peuvent réduire les problèmes d'efficacité de la performance (p. ex., goulots d'étranglement) et réduire le risque projet en évitant les corrections chronophages dans les derniers stades du développement logiciel ou en production.
- Les revues de code, en particulier celles qui mettent l'accent sur l'interaction des bases de données, l'interaction des composants et le traitement des erreurs, peuvent identifier des problèmes d'efficacité de la performances (en particulier en ce qui concerne la logique d'attente et de réessayer et les requêtes inefficaces) et devraient être planifiées au début du cycle de vie du développement logiciel. .
- Le matériel, le logiciel et la bande passante réseau nécessaires à l'exécution des tests de performance doivent être planifiés et budgétisé. Les besoins dépendent principalement de la charge à générer, qui peut être basée sur le nombre d'utilisateurs virtuels à simuler et la quantité de trafic réseau qu'ils sont susceptibles de générer. Si l'on ne tient pas compte de cela, on peut prendre des mesures de rendement non représentatives. Par exemple, la vérification des exigences d'évolutivité d'un site Internet très visité peut nécessiter la simulation de centaines de milliers d'utilisateurs virtuels.
- La génération de la charge requise pour les tests de performance peut avoir une influence significative sur les coûts d'acquisition de matériel et d'outils. Cela doit être pris en compte dans la planification des tests d'efficacité du rendement afin de s'assurer qu'un financement adéquat est disponible.
- Les coûts de génération de la charge pour les tests de performance peuvent être minimisés en louant l'infrastructure de test requise. Il peut s'agir, par exemple, de louer des licences de recharge pour les outils de performance ou d'utiliser les services d'un fournisseur tiers pour

répondre aux besoins matériels (p. ex., services cloud). Si cette approche est adoptée, le temps disponible pour effectuer les tests de performance peut être limité et doit donc être soigneusement planifié.
- Il convient de prendre soin à l'étape de la planification de s'assurer que l'outil de performance à utiliser fournit la compatibilité requise avec les protocoles de communication utilisés par le système sous test.
- Les défauts relatifs à l'efficacité de la performance ont souvent un impact significatif sur le système sous test. Lorsque des exigences d'efficacité de la performance sont impératives, il est souvent utile d'effectuer des tests de performance sur les composants critiques (via les bouchons et les pilotes) afin que les tests puissent commencer tôt dans le cycle de vie au lieu d'attendre les tests système.

Le Syllabus Niveau Fondation Tests de Performance [ISTQB_FLPT_SYL] comprend plus de détails sur la planification des tests de performance.

4.5.3 Spécification des tests de performance

La spécification des tests pour différents types de tests de performance tels que la charge et le stress est basée sur la définition de profils opérationnels. Ils représentent différentes formes de comportements d'utilisateurs lors de leur interaction avec une application.

Le nombre d'utilisateurs par profil opérationnel peut être obtenu à l'aide d'outils de surveillance (lorsque l'application réelle ou une application comparable est déjà disponible) ou en prédisant l'utilisation. Ces prédictions peuvent être basées sur des algorithmes ou fournies par l'organisation métier. Elles sont particulièrement importantes pour préciser le ou les profil(s) opérationnel(s) à utiliser pour les tests d'évolutivité.

Les profils opérationnels sont la base pour le nombre et les types de cas de test à utiliser lors des tests de performance. Ces tests sont souvent contrôlés par des outils de test qui créent des utilisateurs « virtuels » ou simulés en quantités qui représenteront le profil à tester (voir section 6.2.2).

Le Syllabus Niveau Fondation Tests de Performance [ISTQB_FLPT_SYL] comprend plus de détails sur la conception des tests de performance.

4.5.4 Sous-caractéristiques de qualité de l'efficacité de la performance

La classification ISO 25010 des caractéristiques de qualité des produits comprend les sous-caractéristiques suivantes pour l'efficacité de la performance:
- Comportement dans le temps - la capacité d'un composant ou d'un système à répondre aux entrées de l'utilisateur ou du système dans un délai déterminé et dans des conditions précises
- Utilisation des ressources - la capacité du produit logiciel à utiliser les quantités et les types appropriés de ressources
- Capacité - la limite maximale à laquelle un paramètre particulier peut être géré

4.5.4.1 Comportement dans le temps

Le comportement dans le temps se concentre sur la capacité d'un composant ou d'un système à répondre aux entrées de l'utilisateur ou du système dans un délai déterminé et dans des conditions spécifiées. Les mesures du comportement dans le temps varient en fonction des objectifs du test. Pour les composants logiciels individuels, le comportement dans le temps peut être mesuré en fonction des cycles CPU, tandis que pour les systèmes basés sur le client, le comportement dans le temps peut être mesuré en fonction du temps nécessaire pour répondre à une demande particulière de l'utilisateur. Pour les systèmes dont les architectures se composent de plusieurs composants (p. ex., clients, serveurs, bases de données), des mesures du comportement dans le temps sont prises pour les transactions entre les composants individuels afin que les « goulots d'étranglement » puissent être identifiés.

4.5.4.2 Utilisation des ressources

Les tests relatifs à l'utilisation des ressources évaluent l'utilisation des ressources système (p. ex., utilisation de la mémoire, de la capacité du disque, de la bande passante réseau, des connexions) par rapport à un point de référence prédéfini. Celles-ci sont comparées à la fois avec des charges normales et dans des situations de stress, telles que des niveaux élevés de transactions et de volumes de données, afin de déterminer si une croissance anormale de l'utilisation se produit.

Par exemple, pour les systèmes embarqués temps réel, l'utilisation de la mémoire (parfois appelée « empreinte mémoire ») joue un rôle important dans les tests de performance. Si l'empreinte mémoire dépasse la mesure autorisée, le système peut avoir une mémoire insuffisante pour effectuer ses tâches dans les périodes de temps spécifiées. Cela peut ralentir le système ou même conduire à un plantage du système.

L'analyse dynamique peut également être appliquée à l'étude de l'utilisation des ressources (voir la section 3.3.4) et à la détection de goulots d'étranglement de l'efficacité de la performance.

4.5.4.3 Capacité

La capacité d'un système (comprenant logiciel et matériel) représente la limite maximale à laquelle un paramètre particulier peut être géré. Les exigences en capacité sont généralement spécifiées par les parties prenantes techniques et opérationnelles et peuvent se rapporter à des paramètres tels que le nombre maximal d'utilisateurs qui peuvent utiliser une application à un moment donné, le volume maximal de données qui peuvent être transmises par seconde (c.-à-d. la bande passante) et le nombre maximal de transactions qui peuvent être traitées par seconde.

L'approche pour tester les limites de capacité est généralement semblable à l'approche décrite dans les sections 4.5.2 et 4.5.3 pour tester l'efficacité de la performance. Les profils opérationnels pour les tests de capacité se concentrent sur la génération d'une charge qui exerce la limite particulière. Il peut s'agir, par exemple, de générer une charge qui soumet le système à la quantité maximale de transfert de données. Les approches de test de stress et d'évolutivité peuvent également être appliquées à la capacité de tester le comportement du système au-delà des limites de capacité spécifiées (voir les sections 4.5.1.2 et 4.5.1.3 respectivement).

4.6 Tests de maintenabilité

Les logiciels passent souvent une bien plus grande partie de leur vie à être maintenus qu'à être développés. Des tests de maintenance sont effectués pour tester l'impact des changements sur un système opérationnel ou son environnement. Pour s'assurer que la tâche d'effectuer la maintenance est aussi efficace que possible, des tests de maintenance sont effectués pour mesurer la facilité avec laquelle le code peut être analysé, modifié et testé.

Les objectifs typiques des parties prenantes concernées (p.ex., le propriétaire ou l'exploitant du logiciel) pour la maintenabilité, comprennent :

- Minimiser le coût de possession ou d'exploitation du logiciel
- Minimiser le temps d'arrêt requis pour la maintenance logicielle

Des tests de maintenabilité devraient être inclus dans une approche de test où un ou plusieurs des facteurs suivants s'appliquent :

- Des modifications logicielles sont probables après l'entrée en production du logiciel (p. ex., corriger des défauts ou introduire des mises à jour planifiées)
- Les avantages d'atteindre les objectifs de maintenabilité au cours du cycle de vie du développement logiciel sont considérés par les parties prenantes concernées comme

supérieurs aux coûts liés à l'exécution des tests de maintenabilité et à toute modification requise.
- Les risques d'une mauvaise maintenabilité du logiciel (p. ex., long temps de traitement des défauts signalés par les utilisateurs et/ou les clients) justifient la réalisation de tests de maintenabilité

4.6.1 Tests statiques et dynamiques de maintenabilité

Les techniques appropriées pour les tests de maintenabilité comprennent l'analyse statique et les revues telles que discutées dans les sections 3.2 et 5.2. Les tests de maintenabilité doivent être commencés dès que la documentation de conception est disponible et se poursuivre tout au long de l'effort d'implémentation du code. Étant donné que la maintenabilité est intégrée au code et à la documentation pour chaque composant de code individuel, la maintenabilité peut être évaluée dès le début du cycle de vie du développement logiciel sans avoir à attendre un système terminé et en cours d'exécution.

Les tests dynamiques de maintenabilité se concentrent sur les procédures documentées élaborées pour maintenir une application particulière (p. ex., pour effectuer des mises à niveau logicielles). Des sélections de scénarios de maintenance sont utilisées comme cas de test pour s'assurer que les niveaux de service requis sont réalisables avec les procédures documentées. Cette forme de test est particulièrement pertinente lorsque l'infrastructure sous-jacente est complexe et que les procédures de support peuvent impliquer plusieurs département/organisations. Cette forme de test peut avoir lieu dans le cadre des tests d'acceptation opérationnelle.

4.6.2 Sous-caractéristiques de la maintenabilité

La maintenabilité d'un système peut être mesurée en termes d'effort requis pour diagnostiquer les problèmes identifiés dans un système (facilité d'analyse) et tester le système modifié (testabilité). Les facteurs qui influent à la fois sur la facilité d'analyse et la testabilité comprennent l'application de bonnes pratiques de programmation (p. ex., commentaires, nommage des variables, indentation) et la disponibilité de la documentation technique (p. ex., spécifications de conception du système, spécifications de l'interface).

D'autres sous-caractéristiques de qualité pour la maintenabilité [ISO25010] sont :
- Facilité de modification – degré auquel un composant ou un système peut être modifié de façon efficace et efficiente sans introduire de défauts ou dégrader la qualité du produit existant
- Modularité – degré auquel un système, un produit ou un composant est composé de composants distincts de sorte que la modification d'un composant ait un impact minimal sur les autres composants
- Réutilisabilité – degré auquel un élément peut être utilisé dans plus d'un système, ou dans la construction d'autres éléments

4.7 Tests de portabilité

4.7.1 Introduction

Les tests de portabilité relèvent généralement du degré auquel un composant logiciel ou un système peut être transféré dans l'environnement cible, soit directement, soit à partir d'un environnement existant.

[ISO25010] comprend les sous-caractéristiques suivantes de la portabilité :
- Facilité d'installation - la capacité du produit logiciel à être installé dans un environnement spécifié

- Adaptabilité – le degré auquel un composant ou un système peut être adapté à des environnements matériels et logiciels différents ou en évolution
- Facilité de remplacement - la capacité d'un autre produit logiciel à être utilisé à la place du produit logiciel spécifié dans le même but et dans le même environnement

Les tests de portabilité peuvent commencer par les composants individuels (p. ex., la facilité de remplacement d'un composant particulier, comme le remplacement d'un système de gestion de base de données par un autre) puis s'étendre à mesure que plus de code est disponible. La facilité d'installation peut ne pas être testable tant que tous les composants du produit ne fonctionnent pas.

La portabilité doit être conçue et intégrée au produit et doit donc être prise en compte dès les phases de conception et d'architecture. Les revues de l'architecture et de la conception peuvent être particulièrement productives pour cerner les exigences et les problèmes potentiels de portabilité (p. ex., dépendance à l'égard d'un système d'exploitation particulier).

4.7.2 Tests de facilité d'installation

Les tests de facilité d'installation sont effectués sur le logiciel et les procédures écrites utilisées pour installer le logiciel dans son environnement cible. Cela peut inclure, par exemple, le logiciel développé pour installer un système d'exploitation sur un processeur, ou un assistant d'installation (wizard) utilisé pour installer un produit sur un PC client.

Les objectifs typiques des tests de facilité d'installation comprennent :

- Valider que le logiciel peut être installé avec succès en suivant les instructions dans un manuel d'installation (y compris l'exécution de tous les scripts d'installation), ou en utilisant un assistant d'installation. Cela comprend le test des options d'installation pour différentes configurations matérielles/logicielles et pour divers degrés d'installation (p. ex., initiale ou mise à jour).
- Tester si des défaillances se produisant pendant l'installation (p. ex., incapacité à charger des DLL particulières) sont traitées correctement par le logiciel d'installation sans laisser le système dans un état non défini (par exemple, un logiciel partiellement installé ou des configurations système incorrectes)
- Vérifier si une installation/désinstallation partielle peut être effectuée
- Tester si un assistant d'installation peut identifier avec succès des configurations de plate-forme matérielle ou de système d'exploitation invalides
- Mesurer si le processus d'installation peut être terminé en quelques minutes ou après un nombre déterminé d'étapes
- Valider que le logiciel peut être rétrogradé ou désinstallé avec succès

Les tests fonctionnels sont normalement effectués après les tests de facilité d'installation pour détecter les défauts qui pourraient avoir été introduits par l'installation (p. ex., configurations incorrectes, fonctions non disponibles). Les tests d'utilisabilité sont normalement effectués parallèlement aux tests de facilité d'installation (p. ex., pour valider que les utilisateurs reçoivent des instructions compréhensibles et des messages d'information/erreur pendant l'installation).

4.7.3 Tests d'adaptabilité

Les tests d'adaptabilité vérifient si une application donnée peut fonctionner correctement dans tous les environnements cibles prévus (matériel, logiciel, middleware, système d'exploitation, etc.). Un système adaptatif est donc un système ouvert qui est capable d'adapter son comportement en fonction des changements dans son environnement ou dans certaines parties du système lui-même. Pour spécifier les tests d'adaptabilité, il faut que les combinaisons des environnements cibles prévus soient identifiées, configurées et mises à la disposition de l'équipe de test. Ces environnements sont ensuite testés à l'aide d'une sélection de cas de test fonctionnels qui exercent les différents composants présents dans l'environnement.

L'adaptabilité peut se rapporter à la capacité du logiciel à être porté dans divers environnements spécifiés en effectuant une procédure prédéfinie. Les tests peuvent évaluer cette procédure.

Les tests d'adaptabilité peuvent être effectués conjointement avec des tests de facilité d'installation et sont généralement suivis de tests fonctionnels pour détecter les défauts qui pourraient avoir été introduits dans l'adaptation du logiciel à un environnement différent.

4.7.4 Test de facilité de remplacement

Les tests de facilité de remplacement se concentrent sur la capacité des composants logiciels d'un système à être remplacés par d'autres. Cela peut être particulièrement pertinent pour les systèmes qui utilisent des composants sur étagère.

Les tests de facilité de remplacement peuvent être effectués parallèlement à des tests d'intégration fonctionnelle où plus d'un composant alternatif est disponible pour l'intégration dans le système complet. La facilité de remplacement peut être évaluée par revue technique ou inspection aux niveaux de l'architecture et de la conception, où l'accent est mis sur la définition claire des interfaces des composants pouvant être remplacés.

4.8 Test de compatibilité

4.8.1 Introduction

Les tests de compatibilité examinent les aspects suivants [ISO25010]:

- Co-existence – le degré auquel un élément à tester peut fonctionner de manière satisfaisante aux côtés d'autres produits indépendants dans un environnement partagé. Ceci est décrit ci-dessous.
- Interopérabilité – le degré auquel un système échange des informations avec d'autres systèmes ou composants. Cela est décrit dans le syllabus ISTQB Analyste de Test Niveau Avancé [ISTQB_ALTA_SYL].

4.8.2 Tests de coexistence

On dit que des systèmes informatiques qui ne sont pas liés les uns aux autres coexistent lorsqu'ils peuvent fonctionner dans le même environnement (p. ex., sur le même matériel) sans affecter le comportement de l'autre (p. ex., conflits de ressources). Des tests de coexistence devraient être effectués lorsque des logiciels nouveaux ou mis à jour seront déployés dans des environnements qui contiennent déjà des applications installées.

Des problèmes de coexistence peuvent survenir lorsque l'application est testée dans un environnement où elle est la seule application installée (où les problèmes d'incompatibilité ne sont pas détectables) puis déployée sur un autre environnement (par exemple, la production) qui exécute également d'autres applications.

Les objectifs typiques des tests de coexistence comprennent :

- L'évaluation de l'impact négatif possible sur les fonctionnalités lorsque les applications sont chargées dans le même environnement (par exemple, l'utilisation conflictuelle de ressources lorsqu'un serveur exécute plusieurs applications)
- L'évaluation de l'impact sur toute application suite au déploiement de correctifs et de mises à niveau du système d'exploitation

Les problèmes de coexistence devraient être analysés lors de la planification de l'environnement de production ciblé, mais les tests réels sont normalement effectués après que les tests système aient été déroulés avec succès.

5. Revues - 165 mins.

Mots clés
anti-pattern

Objectifs d'apprentissage pour Revues

5.1 Tâches de l'Analyste Technique de Test dans les Revues

TTA 5.1.1 (K2) Expliquer pourquoi la préparation des revues est importante pour l'Analyste Technique de Test

5.2 Utilisation de Checklists dans les Revues

TTA 5.2.1 (K4) Analyser une conception architecturale et identifier les problèmes selon une checklist fournie dans le syllabus

TTA 5.2.2 (K4) Analyser une section de code ou de pseudo-code et identifier les problèmes selon une checklist fournie dans le syllabus

5.1 Tâches de l'Analyste Technique de Test dans les Revues

Les Analystes Techniques de Test doivent participer activement au processus de revue technique, en apportant leur point de vue. Tous les participants à une revue devraient avoir une formation officielle en revue afin de mieux comprendre leurs rôles respectifs. Ils doivent s'impliquer pour que les bénéfices d'une revue technique bien menée soient atteints. Cela comprend le maintien d'une relation de travail constructive avec les auteurs lorsqu'ils décrivent et discutent des commentaires. Pour une description complète des revues techniques, incluant de nombreuses checklists de revue, voir [Wiegers02]. Les Analystes Techniques de Test participent normalement à des revues techniques et à des inspections où ils apportent un point de vue opérationnel (comportemental) qui peut manquer aux développeurs. De plus, les Analystes Techniques de Test jouent un rôle important dans la définition, l'application et la maintenance des checklists de revue et de l'information sur la gravité des défauts.

Quel que soit le type de revue effectué, l'Analyste Technique de Test doit avoir suffisamment de temps pour se préparer. Cela comprend le temps de revue du produit d'activité, le temps de vérifier la cohérence des documents référencés, et le temps de déterminer ce qui pourrait manquer dans le produit d'activité. Sans temps de préparation adéquat, la revue peut devenir un exercice d'édition plutôt qu'une véritable revue. Une bonne revue comprend la compréhension de ce qui est écrit, la détermination de ce qui manque, et la vérification que le produit décrit est compatible avec d'autres produits qui sont déjà développés ou sont en développement. Par exemple, lors de la revue d'un plan de test de niveau d'intégration, l'Analyste Technique de Test doit également tenir compte des éléments qui sont intégrés. Sont-ils prêts pour l'intégration ? Y a-t-il des dépendances qui doivent être documentées ? Existe-t-il des données disponibles pour tester les points d'intégration ? Une revue ne se limite pas au produit d'activités en revue. Elle doit également tenir compte de l'interaction de cet élément avec les autres dans le système.

5.2 Utilisation de Checklists dans les Revues

Les checklists sont utilisées lors des revues pour rappeler aux participants de vérifier des points précis au cours de la revue. Les checklists peuvent également aider à dépersonnaliser la revue, par exemple, « c'est la même checklist que nous utilisons pour chaque revue, nous ne ciblons pas en particulier votre produit d'activités." Les checklists peuvent être génériques et utilisées pour toutes les revues ou axées sur des caractéristiques ou des domaines de qualité spécifiques.
Par exemple, une checklist générique peut vérifier l'utilisation appropriée des termes « doit » et « devrait », vérifier le bon formatage des éléments de même type. Une checklist ciblée peut se concentrer sur des questions de sécurité ou d'efficacité de la performance.

Les checklists les plus utiles sont celles élaborées progressivement par une organisation car elles reflètent :

- La nature du produit
- L'environnement de développement local
  - Personnel
  - Outils
  - Priorités
- L'historique des succès et défauts antérieurs
- Des problèmes particuliers (p. ex., efficacité de la performance, sécurité)

Les checklists devraient être personnalisées pour l'organisation et peut-être pour un projet particulier. Les checklists fournies dans ce chapitre ne sont destinées qu'à servir d'exemples.

Certaines organisations étendent la notion habituelle de checklist logicielle pour inclure des « anti-pattern » qui font référence à des erreurs courantes, à de mauvaises techniques et à d'autres pratiques inefficaces. Le terme dérive du concept populaire de « design-pattern » qui sont des solutions

réutilisables à des problèmes communs qui se sont montrées efficaces dans des situations pratiques [Gamma94]. Un anti-pattern, donc, est une erreur généralement faite, souvent mis en œuvre comme un raccourci.

Il est important de se rappeler que si une exigence n'est pas testable, ce qui signifie qu'elle n'est pas définie de telle sorte que l'Analyste Technique de Test puisse déterminer comment la tester, alors il s'agit d'un défaut. Par exemple, l'exigence « Le logiciel doit être rapide » ne peut pas être testée. Comment l'Analyste Technique de Test peut-il déterminer si le logiciel est rapide ? Si, au lieu de cela, l'exigence disait : « Le logiciel doit fournir un temps de réponse maximal de trois secondes dans des conditions de charge spécifiques », alors la testabilité de cette exigence est sensiblement meilleure en supposant que les « conditions de charge spécifiques » (p. ex., nombre d'utilisateurs simultanés, activités exécutées par les utilisateurs) sont définies. C'est aussi une exigence globale parce que cette seule exigence pourrait facilement engendrer de nombreux cas de test individuels dans une application non triviale. La traçabilité de cette exigence aux cas de test est également essentielle parce que si l'exigence doit changer, tous les cas de test devront être examinés et mis à jour au besoin.

5.2.1 Revues d'architecture

L'architecture logicielle se compose de l'organisation fondamentale d'un système, constitué de ses composants, de leurs relations les uns avec les autres et de son environnement, et des principes régissant sa conception et son évolution. [ISO42010], [Bass03].

Les checklists[1] utilisées pour les revues d'architecture pourraient, par exemple, inclure la vérification de la bonne mise en œuvre des éléments suivants, issus de [Web-2] :

- Pool de connexion - réduire le temps d'exécution associé à l'établissement de connexions à la base de données en établissant un pool partagé de connexions
- Load balancing – répartir la charge uniformément entre un ensemble de ressources
- Traitement distribué
- Mise en cache – utilisation d'une copie locale des données pour réduire le temps d'accès
- Instanciation tardive
- Concurrence des transactions
- Isolement des processus entre Traitement Transactionnel en ligne (Online Transactional Processing (OLTP)) et Traitement Analystique en ligne (Online Analytical Processing (OLAP))
- Réplication des données

5.2.2 Revues de Code

Les checklists pour les revues de code sont nécessairement très détaillées et, comme pour les checklists pour les revues d'architecture, sont plus utiles lorsqu'elles sont spécifiques à un language, à un projet et à une entreprise. L'inclusion d'anti-patterns relatifs au code est utile, en particulier pour les développeurs de logiciels moins expérimentés.

Les checklists[1] utilisées pour les revues de code peuvent inclure les éléments suivants:.

1. Structure
   - Le code implémente-t-il complètement et correctement la conception ?
   - Le code est-il conforme aux normes de codage applicables ?
   - Le code est-il bien structuré, cohérent dans son style et constamment formaté ?
   - Y a-t-il des procédures inutiles, jamais appelées ou un code inaccessible ?
   - Y a t-il des restes de bouchons ou de routines de test dans le code ?

[1] La question de l'examen fournira un sous-ensemble de la liste de contrôle pour répondre à la question

- N'importe quel code peut-il être remplacé par des appels vers des composants réutilisables externes ou des fonctions de bibliothèque ?
- Y at-il des blocs de code répétés qui pourraient être condensés en une seule procédure ?
- L'utilisation du stockage est-elle efficace ?
- Des paramêtres sont-ils utilisés plutôt que des valeurs "en dure" pour des nombres ou des constantes ?
- Certains modules sont-ils excessivement complexes et doivent-ils être restructurés ou divisés en plusieurs modules ?

2. Documentation
- Le code est-il clairement et adéquatement documenté avec un style de commentaires facile à maintenir ?
- Tous les commentaires sont-ils en cohérence avec le code ?
- La documentation est-elle conforme aux normes applicables ?

3. Variables
- Toutes les variables sont-elles correctement définies avec des noms significatifs, cohérents et clairs ?
- Y a-t-il des variables redondantes ou inutilisées ?

4. Opérations arithmétiques
- Le code évite-t-ilde comparer des nombres en virgule flottante ?
- Le code empêche-t-il systématiquement les erreurs d'arrondi ?
- Le code évite-t-il les ajouts et les soustractions sur des nombres dont les magnitudes sont très différentes ?
- Les diviseurs sont-ils testés pour le zéro ?

5. Boucles et branches
- Toutes les boucles, branches et constructions logiques sont-elles complètes, correctes et correctement imbriquées ?
- Les cas les plus courants sont-ils testés en premier dans les chaînes IF-ELSEIF ?
- Tous les cas sont-ils couverts par un bloc IF-ELSEIF ou CASE, y compris les clauses ELSE ou DEFAULT ?
- Chaque instruction CASE a-t-elle un cas par defaut ?
- Les conditions de terminaison de boucle sont-elles évidentes et invariablement réalisables ?
- Les indices ou les sous-scripts sont-ils correctement initialisés, juste avant la boucle ?
- Certaines instructions placées dans des boucles pourraient-elles être placées en dehors des boucles ?
- Le code dans la boucle évite-t-il de manipuler l'indice de la variable ou de l'utiliser à la sortie de la boucle ?

6. Programmation défensive
- Les indices, les pointeurs et les sous-scripts sont-ils testés par rapport aux limites de tableau, d'enregistrement ou de fichier ?
- Les données importées et les arguments d'entrée sont-ils testés pour leur validité et leur exhaustivité ?
- Toutes les variables de sortie sont-elles affectées ?
- L'élément de données correct est-il utilisé dans chaque instruction?
- Chaque allocation de mémoire est-elle libérée ?
- Est-ce que des temporisations et des récupérations d'erreur sont mises en place pour l'accès aux équipements extérieurs ?
- L'existence des fichiers est-elle vérifiée avant d'essayer d'y accéder ?

- Tous les fichiers et périphériques sont-ils laissés dans le bon état à la fin du programme ?

6. Outils de test et automatisation - 180 mins.

Mots clés

capture/rejeux, test piloté par les données, débogage, émulateur, injection de défauts, Lien hypertexte, test dirigés par mots-clés, efficacité de la performance, simulateur, exécution des tests, gestion des tests

Objectifs d'apprentissage pour Outils de test et automatisation

6.1 Définition du Projet d'Automatisation des Tests

TTA-6.1.1 (K2) Résumer les activités que l'Analyste Technique de Test effectue lors de la mise en place d'un projet d'automatisation des tests
TTA-6.1.2 (K2) Résumer les différences entre l'automatisation pilotée par les données et dirigée par mots clés
TTA-6.1.3 (K2) Résumer les problèmes techniques courants qui font que les projets d'automatisation n'atteignent pas le retour sur investissement prévu
TTA-6.1.4 (K3) Construire des mots clés à partir d'un processus métier donné

6.2 Outils de test spécifiques

TTA-6.2.1 (K2) Résumer le but des outils d'insertion de fautes et des outils d'injection de fautes
TTA-6.2.2 (K2) Résumer les principales caractéristiques et les problèmes de mise en œuvre des outils de test de performance
TTA-6.2.3 (K2) Expliquer le but général des outils utilisés pour les tests basés sur le Web
TTA-6.2.4 (K2) Expliquer comment les outils contribuent à la pratique des tests basés sur des modèles
TTA-6.2.5 (K2) Décrire le but des outils utilisés pour soutenir les tests de composants et le processus de build
TTA-6.2.6 (K2) Décrire le but des outils utilisés pour prendre en charge les tests d'applications mobiles

6.1 Définition du Projet d'Automatisation des Tests

Pour être rentables, les outils de test (et en particulier ceux qui permettent de gérer l'exécution des tests) doivent être soigneusement conçus et structurés. La mise en œuvre d'une stratégie d'automatisation de l'exécution des tests sans architecture solide se traduit généralement par un ensemble d'outils coûteux à entretenir, insuffisant et ne permettant pas d'atteindre le retour sur investissement escompté.

Un projet d'automatisation des tests doit être considéré comme un projet de développement de logiciels. Cela comprend le besoin de documentation de l'architecture, de documentation détaillée sur la conception, de revues de conception et de code, de tests de composants et d'intégration de composants, ainsi que de tests finaux du système. Les tests peuvent être inutilement retardés ou compliqués lorsque du code d'automatisation de test instable ou inexact est utilisé.

Il y a plusieurs tâches que l'Analyste Technique de Test peut effectuer pour l'automatisation de l'exécution des tests. Il s'agit notamment de :

- Déterminer qui sera responsable de l'exécution des tests (éventuellement en coordination avec un Test Manager)
- Choisir l'outil approprié pour l'organisation, le calendrier, les compétences de l'équipe et les exigences de maintenance (notez que cela pourrait signifier décider de créer un outil à utiliser plutôt que d'en acquérir un)
- Définir les exigences d'interface entre l'outil d'automatisation et d'autres outils tels que ceux utilisés pour la gestion des tests, la gestion des défauts et ceux utilisés pour l'intégration continue
- Développer tous les adaptateurs qui peuvent être nécessaires pour créer une interface entre l'outil d'exécution de test et le logiciel à tester
- Sélectionner l'approche d'automatisation, c.-à-d. pilotée par mots clés ou dirigées par les données (voir la section 6.1.1 ci-dessous)
- Travailler avec le Test Manager pour estimer le coût de la mise en œuvre, y compris la formation. Dans les projets Agile, cet aspect serait généralement discuté et convenu lors de réunions de planification de projet/sprint avec toute l'équipe.
- Planifier le projet d'automatisation et allouer du temps à la maintenance
- Former des Analystes de Test et des Analystes Métier à l'utilisation et à la fourniture de données pour l'automatisation
- Déterminer comment et quand les tests automatisés seront exécutés
- Déterminer comment les résultats des tests automatisés seront combinés avec les résultats des tests manuels

Dans les projets mettant fortement l'accent sur l'automatisation des tests, un Ingénieur d'Automatisation des Tests peut être chargé de bon nombre de ces activités. (voir le Syllabus Avancé AUtomatisation des Tests) [ISTQB_ALTAE_SYL] pour plus de détails). Certaines tâches organisationnelles peuvent être prises en charge par un Test Manager en fonction des besoins et des préférences du projet. Dans les projets Agile, l'attribution de ces tâches à des rôles est généralement plus souple et moins formelle.

Ces activités et les décisions qui en résulteront influeront sur l'évolutivité et la maintenabilité de la solution d'automatisation. Il faut passer suffisamment de temps à étudier les options, à étudier les outils et les technologies disponibles et à comprendre les projets futurs de l'organisation.

6.1.1 Sélection de l'approche d'automatisation

Cette section tient aborde les facteurs suivants qui influent sur l'approche d'automatisation des tests :

- Automatisation par l'intermédiaire de l'interface graphique utilisateur (GUI)

- Application d'une approche pilotée par les données
- Application d'une approche dirigée par mots clés
- Gestion des défaillances logicielles
- Prise en compte de l'état du système

Le syllabus Niveau Avancé Automatisation des Tests [ISTQB_ALTAE_SYL] détaille la sélection d'une approche d'automatisation.

6.1.1.1 Automatisation par l'intermédiaire de l'interface graphique utilisateur

L'automatisation des tests ne se limite pas aux tests via l'interface graphique utilisateur. Des outils existent pour aider à automatiser les tests au niveau de l'API (Application Programming Interface), via une interface en ligne de commande et d'autres points d'interface dans le logiciel testé. L'une des premières décisions que l'Analyste Technique de Test doit prendre est de déterminer l'interface la plus efficace à utiliser pour automatiser les tests. Les principaux outils d'exécution des tests nécessitent le développement d'adaptateurs pour ces interfaces. La planification doit tenir compte de l'effort de développement de l'adaptateur.

L'une des difficultés des tests à travers l'interface graphique est la tendance de l'interface graphique à changer au fur et à mesure que le logiciel évolue. Selon la façon dont le code d'automatisation des tests est conçu, cela peut entraîner une charge de maintenance importante. Par exemple, l'utilisation de la fonctionnalité de capture/rejeux d'un outil d'automatisation des tests peut entraîner des cas de test automatisés (souvent appelés scripts de test) qui ne s'exécutent plus comme souhaité si l'interface graphique change. Ceci est dû au fait que le script enregistré capture les interactions avec les objets graphiques lorsque le testeur exécute le logiciel manuellement. Si les objets consultés changent, les scripts enregistrés peuvent également avoir besoin d'être mis à jour pour refléter ces modifications.

Les outils de capture/rejeux peuvent être utilisés comme point de départ pratique pour développer des scripts d'automatisation. Le testeur enregistre une session de test et le script enregistré est ensuite modifié pour améliorer la maintenabilité (par ex.., en remplaçant les sections du script enregistré par des fonctions réutilisables).

6.1.1.2 Application d'une approche pilotée par les données

Selon le logiciel testé, les données utilisées pour chaque test peuvent être différentes, bien que les étapes de test exécutées soient pratiquement identiques (p. ex., tester la gestion des erreurs pour un champ d'entrée en entrant plusieurs valeurs invalides et en vérifiant l'erreur retournée pour chacune d'elles). Il est inefficace de développer et de maintenir un script de test automatisé pour chacune de ces valeurs à tester. Une solution technique commune à ce problème est de déplacer les données des scripts vers un fichier ou système externe tel qu'une feuille de calcul ou une base de données. Des fonctions sont écrites pour accéder aux données spécifiques pour chaque exécution du script de test, ce qui permet à un seul script de fonctionner à travers un ensemble de données de test qui fournit les valeurs d'entrée et les valeurs de résultat attendues (par exemple, une valeur affichée dans un champ de texte ou un message d'erreur).Cette approche s'appelle le test automatisé piloté par les données .

Lors de l'utilisation de cette approche, en plus des scripts de test qui traitent les données fournies, un harnais et une infrastructure sont nécessaires pour soutenir l'exécution du script ou de l'ensemble des scripts. Les données réelles définies dans la feuille de calcul ou la base de données sont créées par des analystes de test qui connaissent bien les fonctions métier du logiciel. Dans les projets Agile, le représentant métier (p. ex., Product Owner) peut également participer à la définition de données, en particulier pour les tests d'acceptation. Cette division du travail permet aux responsables du développement de scripts de test (par exemple, l'Analyste Technique de Test) de se concentrer sur l'implémentation de scripts d'automatisation intelligents tandis que l'Analyste de Test conserve la propriété du test réel. Dans la plupart des cas, l'Analyste de Test sera responsable de l'exécution des scripts de test une fois que l'automatisation est implémentée et testée.

6.1.1.3 Application d'une approche dirigée par mots clés

Une autre approche, appelée dirigée par mots clés ou mots actions, va un peu plus loin en séparant également l'action à effectuer sur les données fournies du script de test [Buwalda01]. Afin d'accomplir cette nouvelle séparation, un méta langage de haut niveau qui est descriptif plutôt que directement exécutable est créé. Chaque déclaration de ce langage décrit un processus métier complet ou partiel du domaine qui peut nécessiter des tests. Par exemple, les mots clés du processus métier peuvent inclure « Connexion », «Créer Utilisateur» et « Supprimer Utilisateur». Un mot clé décrit une action de haut niveau qui sera effectuée dans le domaine de l'application. Des actions de plus bas niveau qui dénotent l'interaction avec l'interface logicielle elle-même, telles que : «ClickBouton», «SélectionnerDansListe» ou «DéployerArborescence» peuvent également être définies et peuvent être utilisées pour tester des fonctionnalités d'interface graphique qui ne s'intègrent pas parfaitement dans les mots clés des processus d'entreprise.

Une fois que les mots clés et les données à utiliser ont été définis, l'automaticien de test (p. ex., l'Analyste Technique de Test ou l'ingénieur d'automatisation des tests) traduit les mots clés du processus métier et les actions de niveau inférieur en code d'automatisation des tests. Les mots clés et les actions, ainsi que les données à utiliser, peuvent être stockés dans des feuilles de calcul ou entrés à l'aide d'outils spécifiques qui permettent l'automatisation des tests dirigée par mots clés. Le framework d'automatisation des tests implémente le mot clé sous forme d'un ensemble d'une ou plusieurs fonctions ou scripts exécutables. Les outils lisent les cas de test écrits avec des mots clés et appellent les fonctions de test ou les scripts appropriés qui les implémentent. Les exécutables sont implémentés de manière hautement modulaire pour permettre une cartographie facile vers des mots clés spécifiques. Des compétences en programmation sont nécessaires pour implémenter ces scripts modulaires.

Cette séparation entre la connaissance de la logique métier et les compétences requises en programmation pour implémenter les scripts d'automatisation de test permet d'optimiser l'utilisation des ressources de test. L'Analyste Technique de Test, dans le rôle d'automaticien de test, peut effectivement appliquer des compétences de programmation sans avoir à devenir un expert de domaine dans de nombreux domaines de l'entreprise.

Séparer le code des données changeables permet d'isoler l'automatisation des modifications, d'améliorer la maintenabilité globale du code et d'améliorer le retour sur l'investissement de l'automatisation.

6.1.1.4 Gestion des défaillances logicielles

Dans toute conception d'automatisation de test, il est important d'anticiper et de gérer les défaillances logicielles. En cas de défaillance, l'automaticien de test doit déterminer ce que le logiciel doit faire. La défaillance doit-elle être enregistrée et les tests se poursuivre ? Faut-il mettre fin aux tests ? La défaillance peut-elle être gérée avec une action spécifique (par exemple en cliquant sur un bouton dans une boîte de dialogue) ou peut-être en ajoutant un délai dans le test ? Des défaillances logicielles non gérées peuvent corrompre les résultats des tests ultérieurs et causer un problème avec le test qui s'exécute lorsque la défaillance s'est produite.

6.1.1.5 Prise en compte de l'état du système

Il est également important de tenir compte de l'état du système au début et à la fin des tests. Il peut être nécessaire de s'assurer que le système est retourné à un état prédéfini après la fin de l'exécution du test. Cela permettra d'exécuter une suite de tests automatisés à plusieurs reprises sans intervention manuelle pour réinitialiser le système à un état connu. Pour ce faire, l'automatisation des tests peut devoir, par exemple, supprimer les données qu'elle a créées où modifier l'état des enregistrements dans une base de données. Le framework d'automatisation devrait s'assurer qu'une terminaison appropriée a été effectuée à la fin des tests (c.-à-d. se déconnecter après la fin des tests).

6.1.2 Modélisation des processus métier pour l'automatisation

Afin de mettre en œuvre une approche axée sur les mots clés pour l'automatisation des tests, les processus métier à tester doivent être modélisés dans le langage de mots clés de haut niveau. Il est important que le langage soit intuitif pour ses utilisateurs qui sont susceptibles d'être les Analystes de Test travaillant sur le projet ou, dans le cas des projets Agile, le représentant métier (par exemple, le Product Owner).

Les mots clés sont généralement utilisés pour représenter des interactions métier de haut niveau avec un système. Par exemple, « Annuler_Commande » peut exiger de vérifier l'existence de la commande, de vérifier les droits d'accès de la personne demandant l'annulation, d'afficher l'ordre d'annulation et de demander la confirmation de l'annulation. Des séquences de mots clés (p. ex., « Connexion », « Sélection_Commande », « Annulation_Commande ») et les données de test pertinentes sont utilisées par l'Analyste de Test pour spécifier les cas de test. Ce qui suit est une table d'entrée pour une approche dirigée par mots clés qui pourrait être utilisée pour tester la capacité du logiciel à ajouter, réinitialiser et supprimer des comptes d'utilisateurs :

Mots clés	**Utilisateur**	**Mot de passe**		**Résultat**
Ajouter_Utilisateur	Utilisateur1	Pass1		Message « utilisateur ajouté »
Ajouter_Utilisateur	@Rec34	@Rec35		Message « utilisateur ajouté »
Réinitialiser_MotDePasse	Utilisateur1	Welcome		Message de confirmation de la réinitialisation du mot de passe
Supprimer_Utilisateur	Utilisateur1			Message « Nom d'utilisateur/mot de passe non valide »
Ajouter_Utilisateur	Utilisateur3	Pass3		Message « utilisateur ajouté »
Supprimer_Utilisateur	Utilisateur2			Message « utilisateur non trouvé »

Le script d'automatisation qui utilise cette table rechercherait les valeurs d'entrée à utiliser par le script d'automatisation. Par exemple, lorsqu'il arrive à la ligne avec le mot clé «Supprimer_Utilisateur»,seul le nom d'utilisateur est requis. Pour ajouter un nouvel utilisateur, le nom d'utilisateur et le mot de passe sont nécessaires. Les valeurs d'entrée peuvent également être référencées à partir d'un magasin de données comme indiqué avec le deuxième mot clé « Ajouter _Utilisateur» où une référence aux données est saisie plutôt que les données elles-mêmes offrant plus de flexibilité pour accéder aux données qui peuvent changer au fur et à mesure que les tests s'exécutent. Cela permet de combiner des techniques pilotées par les données avec le schéma de mots clés.

Certaines problématiques sont à prendre en considération :

- Plus les mots clés sont granulaires, plus les scénarios qui peuvent être couverts sont spécifiques, mais le langage de haut niveau peut devenir plus complexe à maintenir.
- Permettre aux Analystes de Test de spécifier des actions de bas niveau «ClickBouton», «SélectionnerDansListe», etc.) simplifie la gestion de différentes situations. Toutefois, étant donné que ces actions sont directement liées à l'interface graphique, les tests peuvent également nécessiter plus de maintenance en cas de changements.
- L'utilisation de mots clés agrégés peut simplifier le développement mais compliquer la maintenance. Par exemple, il peut y avoir six mots clés différents qui créent collectivement un enregistrement. Un seul mot clé appelant les six mots clés consécutifs devrait-il être créé pour simplifier cette action ?

- Quel que soit le temps initialement passé dans l'analyse initiale des mots clés, il y aura souvent des moments où de nouveaux mots clés ou des modifications de mots clés seront nécessaires. Un mot clé a deux dimensions (la logique métier qu'il traduit et la fonctionnalité d'automatisation pour l'exécuter). Par conséquent, un processus doit être créé pour traiter des deux dimensions.

L'automatisation des tests par mots clés peut réduire considérablement les coûts de maintenance de l'automatisation des tests, mais elle est plus coûteuse, plus difficile à développer et nécessite un temps de conception plus important afin d'obtenir le retour sur investissement attendu.

Le syllabus Avancé Automatisation des Tests [ISTQB_ALTAE_SYL] comprend plus de détails sur la modélisation des processus métier pour l'automatisation.

6.2 Outils de Test Spécifiques

Cette section apporte des informations générales sur les outils susceptibles d'être utilisés par un Analyste Technique de Test au-delà de ce qui est discuté dans le Syllabus Niveau [ISTQB_FL_SYL].

Des informations détaillées sur les outils sont fournies par différents Syllabi ISTQB® :
- Test d'Applications Mobiles [ISTQB_FLMAT_SYL]
- Test de Performance [ISTQB_FLPT_SYL]
- Model-Based Testing [ISTQB_FLMBT_SYL]
- Automatisation des Tests [ISTQB_ALTAE_SYL]

6.2.1 Outils Insertion de Fautes/Injection de Fautes

Les outils d'insertion de fautes modifient en fait le code testé (éventuellement à l'aide d'algorithmes prédéfinis) afin de vérifier la couverture obtenue par des tests spécifiques. Lorsqu'ils sont utilisés de manière systématique, cela permet d'évaluer la qualité des tests (c'est-à-dire leur capacité à détecter les défauts insérés) et, le cas échéant, de les améliorer.

Les outils d'injection de fautes fournissent délibérément des entrées incorrectes au logiciel pour s'assurer que celui-ci pourra les gérer correctement. Les entrées sont injectées pour perturber le flot d'exécution normal du code et permettre d'étendre la couverture du test (par exemple, pour couvrir des conditions de test plus négatives et pour tester des mécanismes de traitement des erreurs).

Ces deux types d'outils sont généralement utilisés par l'Analyste Technique de Test, mais peuvent également être utilisés par le développeur lors du test du code nouvellement développé.

6.2.2 Outils de test de performance

Les principales fonctions d'un outil de test de performance sont les suivantes :
- Génération de charge
- Mesure, surveillance, visualisation et analyse de la réponse du système à une charge donnée
- Information sur le comportement des ressources des composants système et réseau

La génération de charge est effectuée par la mise en œuvre d'un profil opérationnel prédéfini (voir Section 4. 5.3) comme script. Le script peut d'abord être capturé pour un seul utilisateur (éventuellement à l'aide d'un outil de capture/rejeux) et est ensuite implémenté pour le profil opérationnel spécifié à l'aide de l'outil de test de performance. Cette implémentation doit tenir compte de la variation des données par transaction (ou ensembles de transactions).

Les outils de performance génèrent une charge en simulant un grand nombre d'utilisateurs simultanés (utilisateurs « virtuels ») désignés pour accomplir des tâches suivant leurs profils opérationnels, ainsi que des volumes spécifiques de données d'entrée. Par rapport aux scripts d'exécution automatisée de

test individuels, de nombreux scripts de test de performances reproduisent l'interaction utilisateur avec le système au niveau du protocole de communication et non en simulant l'interaction utilisateur via une interface utilisateur graphique. Cela réduit généralement le nombre de « sessions » distinctes nécessaires pendant le test. Certains outils de génération de charge peuvent également solliciter l'application à partir de son interface utilisateur pour mesurer plus étroitement le temps de réponse pendant que le système est sous charge.

Un large éventail de mesures sont prises par un outil de test de performance pour permettre l'analyse pendant ou après l'exécution du test. Les mesures prises et les rapports fournis peuvent comprendre les informations suivantes :

- Nombre d'utilisateurs simulés tout au long du test
- Nombre et type de transactions générées par les utilisateurs simulés et taux d'aboutissement des transactions
- Temps de réponse à des demandes de transaction particulières faites par les utilisateurs
- Rapports et graphiques mettant en relation les temps de réponse et la charge
- Rapports sur l'utilisation des ressources (p. ex., utilisation au fil du temps avec des valeurs minimum et maximum)

Les facteurs importants à prendre en compte dans la mise en œuvre des outils de test de performance comprennent :

- Le matériel et la bande passante réseau nécessaires pour générer la charge
- La compatibilité de l'outil avec le protocole de communication utilisé par le système sous test
- La flexibilité de l'outil pour implémenter facilement différents profils opérationnels
- Les fonctions de surveillance, d'analyse et de production de rapports requises

Les outils de test de performance sont généralement acquis plutôt que développés en interne en raison de l'effort requis pour les développer. Il peut toutefois être approprié de développer un outil de performance spécifique si des restrictions techniques empêchent l'utilisation d'un produit disponible, ou si le profil de charge et les installations à fournir sont relativement simples. De plus amples détails sur les outils de test de performance sont fournis dans le Syllabus sur les Tests de Performance au niveau Fondation [ISTQB_FLPT_SYL].

6.2.3 Outils pour les tests Web

Une variété d'outils spécialisés open source et commerciaux sont disponibles pour les tests web. La liste suivante montre le but de certains des outils de test web courants :

- Outils de test d'hyperliens sont utilisés pour parcourir les pages et vérifier qu'aucun hyperlien cassé ou manquant n'est présent sur un site Web
- Vérificateurs HTML et XML sont des outils qui vérifient la conformité aux normes HTML et XML des pages créées par un site Web
- Simulateurs de charge pour tester la façon dont le serveur réagira lorsqu'un grand nombre d'utilisateurs se connectent
- Outils d'exécution d'automatisation légers qui fonctionnent avec différents navigateurs
- Outils pour scanner le serveur, vérifier des fichiers orphelins (non liés)
- Vérificateurs de la syntaxe HTML
- Vérificateurs des fichiers CSS (Cascading Style Sheet)
- Outils pour vérifier les infractions aux normes, p. ex., normes d'accessibilité en vertu de l'article 508 des Etats-Unis ou M/376 en Europe
- Outils qui identifient différents problèmes de sécurité

Voici de bonnes sources d'outils de test web open source

- Le "World Wide Web Consortium" (W3C) [Web-3] Cette organisation établit des normes pour Internet et fournit une variété d'outils pour vérifier les erreurs par rapport à ces normes.

- Le “Web Hypertext Application Technology Working Group” (WHATWG) [Web-5]. Cette organisation établit des normes HTML. Ils ont un outil qui effectue la validation HTML [Web-6].

Certains outils qui incluent un moteur de cartographie web peuvent également fournir des informations sur la taille des pages et sur le temps nécessaire pour les télécharger,et sur la présence ou non d’une page (par exemple, l’erreur HTTP 404). Cela fournit des informations utiles pour le développeur, le webmaster et le testeur.

Les Analystes de Test et les Analystes Techniques de Test utilisent ces outils principalement lors des tests système.

6.2.4 Outils de Tests Basés sur des Modèles

Le test basé sur des modèles (Model-Based Testing - MBT) est une technique par laquelle un modèle formel tel qu’une machine à état fini est utilisé pour décrire le comportement prévu à l’exécution d’un système contrôlé par un logiciel. Les outils MBT commerciaux (voir [Utting07]) fournissent souvent un moteur qui permet à un utilisateur d'exécuter » le modèle. Des chemins d’exécution particuliers peuvent être enregistrés et utilisés comme cas de test. D’autres modèles exécutables tels que les réseaux de Petri et les diagrammes d’états sont également utilisés dans une approche MBT.

Les modèles MBT (et les outils) peuvent être utilisés pour générer de grands ensembles de chemins d’exécution distincts. Les outils MBT peuvent également aider à réduire le très grand nombre de chemins possibles qui peuvent être générés dans un modèle. Les tests à l’aide de ces outils peuvent fournir une vue différente du logiciel à tester. Cela peut entraîner la découverte de défauts qui auraient pu être manqués par des tests fonctionnels.

De plus amples détails sur les outils de test basés sur des modèles sont fournis dans le Syllabus Model Based Testing [ISTQB_FLMBT_SYL].

6.2.5 Outils de Test de Composants et de Build

Bien que les outils de test de composants et d’automatisation de Build soient des outils de développement, dans de nombreux cas, ils sont utilisés et entretenus par les Analystes Techniques de Test, en particulier dans le contexte du développement d’Agile.

Les outils de test de composants sont souvent spécifiques au langage utilisé pour programmer un module. Par exemple, si Java a été utilisé comme langage de programmation, JUnit peut être utilisé pour automatiser les tests unitaires. Beaucoup d’autres langages ont leurs propres outils de test ; ceux-ci sont collectivement appelés frameworks xUnit. Un tel framework génère des objets de test pour chaque classe créée, simplifiant ainsi les tâches que le programmeur doit effectuer lors de l’automatisation des tests de composants.

Les outils de débogage facilitent le test manuel des composants à un niveau très bas, en permettant aux développeurs et aux Analystes Techniques de Test de modifier les valeurs des variables pendant l’exécution du code et de parcourir le code ligne par ligne pendant les tests. Les outils de debugging sont également utilisés pour aider le développeur à isoler et identifier les problèmes dans le code lorsqu’une défaillance est signalée par l’équipe de test.

Les outils d’automatisation de build permettent de déclencher automatiquement un nouveau build chaque fois qu’un composant est changé. Une fois le build terminé, d’autres outils exécutent automatiquement les tests de composants. Ce niveau d’automatisation autour du processus de build est généralement observé dans un environnement d’intégration continue.

Lorsqu’il est mis en place correctement, cet ensemble d’outils peut avoir un effet très positif sur la qualité des builds livrés au test. Si une modification apportée par un programmeur introduit des défauts de

régression dans le build, cela provoquera généralement l'échec de certains des tests automatisés, déclenchant une investigation immédiate sur la cause des défaillances avant que la build ne soit livré dans l'environnement de test.

6.2.6 Outils de Test d'Applications Mobiles

Des émulateurs et des simulateurs sont fréquemment utilisés pour soutenir les tests d'applications mobiles.

6.2.6.1 Simulateurs

Un simulateur mobile modélise l'environnement l'environnement d'exécution de la plate-forme mobile. Les applications testées sur un simulateur sont compilées dans une version dédiée, qui fonctionne sur le simulateur mais pas sur un véritable appareil. Les simulateurs sont parfois utilisés en remplacement des appareils réels dans les tests. Toutefois, l'application testée sur un simulateur diffère de l'application qui sera déployée.

6.2.6.2 Emulateurs

Un émulateur mobile modélise le matériel et utilise le même environnement d'exécution que le matériel physique. Une application compilée pour être déployée et testée sur un émulateur pourrait également être utilisée par le mobile réel.

Les émulateurs sont souvent utilisés pour réduire le coût des environnements de test en remplaçant les appareils réels. Toutefois, un émulateur ne peut pas remplacer complètement un appareil parce que l'émulateur peut se comporter d'une manière différente de l'appareil mobile qu'il tente d'imiter. En outre, certaines fonctionnalités peuvent ne pas être prises en charge telles que les aspect tactiles, l'accéléromètre, et d'autres. Ceci est en partie causé par les limitations de la plate-forme utilisée pour exécuter l'émulateur.

6.2.6.3 Aspects communs

Les simulateurs et émulateurs sont utiles dans les premiers stades du développement car ceux-ci s'intègrent généralement aux environnements de développement et permettent un déploiement rapide, des tests et un suivi des applications. L'utilisation d'un émulateur ou d'un simulateur nécessite de le lancer, d'installer l'application nécessaire dessus, puis de tester l'application comme si elle se trouvait sur l'appareil réel. Chaque environnement de développement de système d'exploitation mobile est généralement livré avec son propre émulateur et simulateur. Des émulateurs et simulateurs tiers sont également disponibles.

Habituellement émulateurs et simulateurs permettent le réglage de divers paramètres d'utilisation. Ces paramètres peuvent inclure l'émulation réseau à différentes vitesses, les forces du signal et les pertes de paquets, le changement d'orientation, la génération d'interruptions et les données de localisation GPS. Certains de ces paramètres peuvent être très utiles parce qu'ils peuvent être difficiles ou coûteux à reproduire avec des appareils réels, notamment les positions GPS ou les forces du signal.

Le Syllabus NIveau Fondation sur les Tests d'Application Mobile [ISTQB_FLMAT_SYL] comprend plus de détails.

7. Références

7.1 Standards

Les normes suivantes sont mentionnées dans ces chapitres respectifs.

- [RTCA DO-178C/ED-12C]: Software Considerations in Airborne Systems and Equipment Certification, RTCA/EUROCAE ED12C. 2013.
  Chapter 2

- [ISO9126] ISO/IEC 9126-1:2001, Software Engineering – Software Product Quality
  Chapter 4

- [ISO25010] ISO/IEC 25010 (2014) Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) System and software quality models
  Chapters 2 and 4

- [ISO29119] ISO/IEC/IEEE 29119-4 International Standard for Software and Systems Engineering - Software Testing Part 4: Test techniques. 2015
  Chapter 2

- [ISO42010] ISO/IEC/IEEE 42010:2011
  Systems and software engineering - Architecture description
  Chapter 5

- [IEC61508] IEC 61508-5 (2010) Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems, Part 5: Examples of methods for the determination of safety integrity levels
  Chapter 2

7.2 Documents ISTQB® (versions originales en anglaise)

- [ISTQB_AL_OVIEW] Advanced Level Overview, Version 2019
- [ISTQB_ALSEC_SYL] Advanced Level Security Testing Syllabus, Version 2016
- [ISTQB_ALTAE_SYL] Advanced Level Test Automation Engineer Syllabus, Version 2017
- [ISTQB_FL_SYL] Foundation Level Syllabus, Version 2018
- [ISTQB_FLPT_SYL] Foundation Level Performance Testing Syllabus, Version 2018
- [ISTQB_FLMBT_SYL] Foundation Level Model-Based Testing Syllabus, Version 2015
- [ISTQB_ALTA_SYL] Advanced Level Test Analyst Syllabus, Version 2019
- [ISTQB_ALTM_SYL] Advanced Level Test Manager Syllabus, Version 2012
- [ISTQB_FLMAT_SYL] Foundation Level Mobile Application Testing Syllabus, 2019
- [ISTQB_GLOSSARY] Glossary of Terms used in Software Testing, Version 3.2, 2019

Les versions françaises disponibles se trouvent sur www.cftl.fr

7.3 Livres

[Bass03] Len Bass, Paul Clements, Rick Kazman “Software Architecture in Practice (2nd edition)”, Addison-Wesley 2003, ISBN 0-321-15495-9

[Bath14] Graham Bath, Judy McKay, “The Software Test Engineer’s Handbook (2nd edition)”, Rocky Nook, 2014, ISBN 978-1-933952-24-6

[Beizer90] Boris Beizer, "Software Testing Techniques Second Edition", International Thomson Computer Press, 1990, ISBN 1-8503-2880-3
[Beizer95] Boris Beizer, “Black-box Testing”, John Wiley & Sons, 1995, ISBN 0-471-12094-4
[Burns18] Brendan Burns, “Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services”, O’Reilly, 2018, ISBN 13: 978-1491983645
[Buwalda01]: Hans Buwalda, “Integrated Test Design and Automation”, Addison-Wesley Longman, 2001, ISBN 0-201-73725-6
[Copeland03]: Lee Copeland, “A Practitioner's Guide to Software Test Design”, Artech House, 2003, ISBN 1-58053-791-X
[Gamma94] Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1994, ISBN 0-201-63361-2
[Jorgensen07]: Paul C. Jorgensen, “Software Testing, a Craftsman’s Approach third edition”, CRC press, 2007, ISBN-13:978-0-8493-7475-3
[Kaner02]: Cem Kaner, James Bach, Bret Pettichord; “Lessons Learned in Software Testing”; Wiley, 2002, ISBN: 0-471-08112-4
[Koomen06]: Tim Koomen, Leo van der Aalst, Bart Broekman, Michael Vroon, "TMap Next for result-driven testing"; UTN Publishers, 2006, ISBN: 90-72194-79-9
[McCabe76] Thomas J. McCabe, "A Complexity Measure", IEEE Transactions on Software Engineering, Vol. SE-2, No. 4, December 1976. PP 308-320
[McCabe96] Arthur H. Watson and Thomas J. McCabe. "Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric" (PDF), 1996, NIST Special Publication 500-235.
[NIST96] Arthur H. Watson and Thomas J. McCabe, "Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric", NIST Special Publication 500-235, Prepared under NIST Contract 43NANB517266, September 1996.
[Splaine01]: Steven Splaine, Stefan P. Jaskiel, “The Web-Testing Handbook”, STQE Publishing, 2001, ISBN 0-970-43630-0
[Utting07] Mark Utting, Bruno Legeard, “Practical Model-Based Testing: A Tools Approach”, Morgan-Kaufmann, 2007, ISBN: 978-0-12-372501-1
[Whittaker04]: James Whittaker and Herbert Thompson, “How to Break Software Security”, Pearson / Addison-Wesley, 2004, ISBN 0-321-19433-0
[Wiegers02] Karl Wiegers, "Peer Reviews in Software: A Practical Guide", Addison-Wesley, 2002, ISBN 0-201-73485-0

7.4 Autres références

Les références suivantes indiquent des informations disponibles sur Internet. Même si ces références ont été vérifiées au moment de la publication de ce syllabus de niveau avancé, l’ISTQB® ne peut être tenu responsable si les références ne sont plus disponibles.

[Web-1] http://www.nist.gov NIST National Institute of Standards and Technology,
[Web-2] http://www.codeproject.com/KB/architecture/SWArchitectureReview.aspx
[Web-3] http://www.W3C.org
[Web-4] https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project
[Web-5] https://whatwg.org
[Web-6] https://whatwg.org/validator/

Chapitre 4: [Web-1] [Web-4]
Chapitre 5: [Web-2]
Chapitre 6: [Web-3] [Web-5] [Web-6]

8. Appendix A : Vue d'ensemble des caractéristiques de qualité

Le tableau suivant compare les caractéristiques de qualité décrites dans l'ISO 9126 (tel qu'il est utilisé dans la version 2012 du syllabus Analyste Technique de Test) avec celles du nouveau [ISO25010] (tel qu'utilisé dans la version 2019 du Syllabus). Seules les caractéristiques pertinentes pour l'Analyste Technique de Test.

ISO/IEC 25010	ISO/IEC 9126-1	Notes
Efficacité de la performance	**Efficacité**	
Comportement dans le temps	Comportement dans le temps	
Utilisation des ressources	Utilisation des ressources	
Capacité		Nouvelle sous-caractéristique
Compatibility		Nouvelle caractéristique
Co-existence	Co-existence	Déplacé de portabilité
Interoperability		Déplacé de fonctionnalité (Analyste de Test)
Fiabilité	**Fiabilité**	
Maturité	Maturité	
Disponibilité		Nouvelle sous-caractéristique
Tolérance aux fautes	Tolérance aux fautes	
Récupération	Récupération	
Security	**Sécurité**	Pas de sous-marinprécédent - caractéristiques
Confidentialité		Nouvelle sous-caractéristique
Intégrité		Nouvelle sous-caractéristique
Non-répudiation		Nouvelle sous-caractéristique
Responsabilité		Nouvelle sous-caractéristique
Authenticité		Nouvelle sous-caractéristique
Maintenabilité	**Maintenabilité**	
Modularité		Nouvelle sous-caractéristique
Réutilisabilité		Nouvelle sous-caractéristique
Facilité d'analyse	Facilité d'analyse	
Facilité de modification	Stabilité	Combine Facilité de changement et Stabilité
	Facilité de changement	
Testabilité	Testabilité	
Portabilité	**Portabilité**	
Adaptabilité	Adaptabilité	
Facilité d'installation	Facilité d'installation	
	Co-existence	Changé en compatibilité

Facilité de remplacement	Facilité de remplacement	
	Compatibilité	Supprimé en 25010

9. Index