Testeur certifié niveau avancé
Analyste Technique de Test
(CTAL-TTA)
Syllabus

V4.0

International Software Testing Qualifications Board

Copyright

Traduction française : Olivier DENOO, Eric RIOU du COSQUER, Bruno LEGEARD et Jean-Baptiste CROUIGNEAU

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
2021 v4.0 Alpha	07/12/2020	Version pour revue Alpha : • Améliorer le texte • Supprimer la sous-section associée au K3 TTA-2.6.1 (2.6 Test des Chemins) et supprimer le LO • Supprimer la sous-section associée au K2 TTA-3.2.4 (3.2.4 Graphes d'appel) et supprimer le LO • Réécrire la sous-section associée à TTA-3.2.2 (3.2.2 Analyse des flots de données) et en faire un K3 • Réécrire les sections associées à TTA-4.4.1 et TTA-4.4.2 (4.4. Test de fiabilité) • Réécrire les sections associées à TTA-4.5.1 et TTA-4.5.2 (4.5 Test de la performance) • Ajouter la section 4.9 sur les Profils opérationnels. • Réécrire la section associée à TTA-2.8.1 (2.7 Sélectionner une Technique de Test Boîte-Blanche) • Réécrire TTA-3.2.1 pour inclure la complexité cyclomatique (pas d'impact sur les questions d'examen) • Réécrire TTA-2.4.1 (MC/DC) pour la rendre cohérente avec les autres Los boite blanche (pas d'impact sur les questions d'examen)
2021 v4.0 Beta	01/03/2021	Mise à jour sur la base des retours de la revue Alpha
v4.0	28/04/2021	Mise à jour sur la base des retours de la revue Beta
v4.0 FR	11/11/2021	Version française

Table des Matières

Remerciements

La version 2019 de ce document a été produite 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

Ce document a été produit par une équipe de l'International Software Testing Qualifications Board, Groupe de travail niveau avancé : Armin Born, Adam Roman, Stuart Reid.

La version mise à jour v4.0 de ce document a été produite par une équipe principale de l'International Software Testing Qualifications Board, Groupe de travail niveau avancé : Armin Born, Adam Roman, Christian Graf, Stuart Reid.

Les personnes suivantes ont participé à la revue, aux commentaires et aux choix d'évolutions de la version v4.0 de ce syllabus :

Ágota Horváth	Lloyd Roden	Paul Weymouth
Benjamin Timmermans	Matthias Hamburg	Péter Földházi Jr.
Erwin Engelsma	Meile Posthuma	Rik Marselis
Gary Mogyorodi	Nishan Portoyan	Tal Pe'er
Geng Chen	Joan Killeen	Vécsey-Juhász Adél
Gergely Ágnecz	Ole Chr. Hansen	Wang Lijuan
Jane Nash	Pálma Polyák	Zuo Zhenlei

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 30 juin 2021.

0. Introduction à ce syllabus

0.1 Objectif de ce syllabus

Ce syllabus constitue la base pour le niveau avancé Analyste Technique 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 Analyste Technique de Test de l'ISTQB® est un document séparé [ISTQB_AL_TTA_OVIEW] qui contient les informations suivantes :

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

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 doit être obtenu.

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, utilisant n'importe quel cycle de développement logiciel. Le syllabus ne contient pas d'objectif d'apprentissage spécifique à un modèle 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 20 heures 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 (300 minutes)
- Chapitre 3 : Analyses Statique et Dynamique (180 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, risque projet, 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 demandera 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. Ils devraient également contribuer à l'identification et au traitement des risques projet associés aux environnements de test, tels que l'acquisition et la mise en place d'environnements de test pour les tests de performance, de fiabilité et de sécurité.

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 leurs expériences 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, les experts techniques 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 de ce syllabus.

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.

La probabilité d'un risque produit est généralement interprétée comme la probabilité de la défaillance dans le système testé. L'Analyste Technique de Test contribue à la recherche et à la compréhension de la probabilité de chaque risque produit, tandis que l'Analyste de Test contribue à comprendre l'impact métier du problème s'il se produit.

Les risques projet qui deviennent des problèmes peuvent avoir une incidence sur le succès global du projet. En règle générale, les facteurs de 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 élevé des exigences techniques

Les risques produit qui deviennent des problèmes peuvent entraîner un plus grand nombre de défauts. En règle générale, les facteurs de risques produit génériques suivants doivent être considérés :

- Complexité de la technologie
- Complexité du code
- Quantité de changements dans le code source (insertions, suppressions, modifications)
- 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 probabilité du risque initial conformément aux lignes directrices établies par le Test Manager. La valeur initiale peut être modifiée par le Test Manager lorsque tous les points de vue des parties prenantes ont été considérés. L'impact du risque est normalement déterminé par l'Analyste de Test.

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 :

- Concevoir des cas de test pour ces risques en abordant des zones à haut risque et aider à évaluer le risque résiduel
- Réduire les risques en exécutant les tests conçus 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. Des informations supplémentaires peuvent être obtenues avec des Syllabi Spécialistes de l'ISTQB ®, tel que le syllabus sur les tests de sécurité [CTSL_SEC_SYL] et le syllabus sur les tests de performance [CTSL_PT_SYL].

2. Techniques de Test Boîte-Blanche - 300 mins.

Mots clés
Test d'API, condition atomique, flot de contrôle, test des décisions, test des conditions/décisions modifiées, test des conditions multiples, niveau d'intégrité de sureté, test des instructions, technique de test boîte-blanche

Objectifs d'apprentissage pour Techniques de Test Boîte-Blanche

2.2 Test des Instructions
TTA-2.2.1 (K3) Concevoir des cas de test pour un objet de test donné en appliquant le test des instructions pour atteindre un niveau défini de couverture

2.3 Test des Décisions
TTA-2.3.1 (K3) Concevoir des cas de test pour un objet de test 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) Concevoir des cas de test pour un objet de test donné en appliquant la technique de test des conditions/décisions modifiées pour atteindre une couverture complète des conditions/décisions modifiées (MC/DC)

2.5 Test des Conditions Multiples
TTA-2.5.1 (K3) Concevoir des cas de test pour un objet de test donné en appliquant la technique de test des conditions multiples pour atteindre un niveau défini de couverture

2.6 Test des Chemins *(a été supprimé de la version v4.0 de ce syllabus)*
TTA-2.6.1 a été supprimé de la version v4.0 de ce syllabus

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 avec un flot de contrôle, 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. Les cas de test générés par les techniques satisfont des critères de couverture qui sont définis comme des objectifs et sont mesurés. Atteindre une couverture complète (par exemple 100%) ne signifie pas que l'ensemble des tests est terminé, mais plutôt que la technique utilisée ne suggère plus de tests utiles supplémentaires pour la structure à l'étude.

Les entrées de test sont générées pour s'assurer qu'un cas de test exerce une partie particulière du code (par exemple, une instruction ou un résultat de décision). Déterminer les entrées de test qui entraîneront l'exercice d'une partie particulière du code peut être difficile, surtout si la partie du code à exercer se trouve à la fin d'un sous-chemin de flux de contrôle long avec plusieurs décisions à prendre. Les résultats attendus sont identifiés en fonction d'une source externe à cette structure, telle qu'une exigence ou une spécification de conception ou une autre base de test.

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 d'API

La Syllabus Fondation [ISTQB_FL_SYL] introduit le test des instructions et le test des décisions. Le test des instructions se concentre sur l'exécution des instructions exécutables dans le code, alors que le test des décisions exerce les résultats des décisions.

Les techniques des Conditions/Décisions Modifiées et des conditions multiples listées ci-dessus sont basées sur des prédicats de décision contenant de multiples conditions et trouvent le même type de défauts. Quelle que soit la complexité d'une décision, elle sera évaluée 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.

Se référer à [ISO 29119] pour plus de détails sur la spécification, et des exemples de ces techniques.

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
Atteindre un couverture complète des instructions devrait être considéré comme un minimum pour tout code testé, bien que cela ne soit pas toujours possible en pratique.

Limitations/Difficultés
Atteindre un couverture complète des instructions devrait être considéré comme un minimum pour tout code testé, bien que cela ne soit pas toujours possible en pratique à cause de contraintes liées au temps et/ou à l'effort disponible. Même des pourcentages élevés de couverture des instructions peuvent ne

pas détecter certains défauts dans la logique du code. Dans de nombreux cas, il n'est pas possible d'obtenir une couverture des instructions de 100 % en raison d'un code inaccessible. Bien que le code inaccessible ne soit généralement pas considéré comme une bonne pratique de programmation, il peut exister, par exemple, si une instruction switch doit avoir un cas par défaut, mais que tous les cas possibles sont traités explicitement.

2.3 Test des décisions

Le test des décisions exerce le résultat des décisions dans le code. 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, il y a un flot de contrôle pour la sortie VRAI et un pour la sortie FAUX; pour une instruction CASE; il peut y avoir plusieurs sorties possibles; pour une instruction LOOP, il existe un flot de contrôle pour le résultat VRAI de la condition de boucle et un pour le résultat FAUX).

La couverture est mesurée comme le nombre de résultats de décision exercés par les tests divisé par le nombre de résultats de décision dans l'objet de test, et est normalement exprimée en pourcentage. Notez qu'un seul cas de test peut exercer plusieurs résultats de décision.

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 uniquement les sorties VRAI et FAUX, sans tenir compte de la complexité de sa structure interne.

Le test des branches est souvent utilisé de manière interchangeable avec le test des décisions, car couvrir toutes les branches et couvrir tous les résultats de décision peut être obtenu avec les mêmes tests. Le test de branche exerce les branches dans le code, où une branche est normalement considérée comme un arc du graphe de flot de contrôle. Pour les programmes sans décision, la définition de la couverture des décisions ci-dessus donne une couverture de 0/0, qui n'est pas définie, quel que soit le nombre de tests exécutés, tandis que la branche unique du point d'entrée au point de sortie (en supposant un point d'entrée et de sortie) entraînera une couverture de branche de 100%. Pour combler cette différence entre les deux mesures, l'ISO 29119-4 exige qu'au moins un test soit exécuté sur du code sans aucune décision pour atteindre une couverture décisionnelle de 100 %, ce qui rend la couverture des décisions de 100 % et la couverture des branches de 100 % équivalentes pour presque tous les programmes. De nombreux outils de test qui fournissent des mesures de couverture, y compris ceux utilisés pour tester les systèmes liés à la sécurité, utilisent une approche similaire.

Applicabilité
Ce niveau de couverture doit être pris en compte lorsque le code testé est à un niveau important de risque, voire critique (voir les tableaux de la section 2.7.2 pour les systèmes liés à la sécurité). Cette technique peut être utilisée pour le code et pour tout modèle impliquant des points de décision, tels que les modèles de processus métier.

Limitations/Difficulties
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 du résultat de ces conditions.

2.4 Test des Conditions/Décisions Modifiées

Par rapport au test des décisions, qui considère la décision complète comme un tout et évalue les résultats VRAI et FAUX, le test des Conditions/Décisions Modifiées considère la façon dont une décision est structurée lorsqu'elle comporte plusieurs conditions (si une décision est composée d'une seule condition atomique, il s'agit simplement du test des décisions).

Chaque prédicat de décision est composé d'une ou plusieurs conditions atomiques, chacune d'elles étant évaluée comme une valeur bouléenne. Celles-ci sont logiquement combinées pour déterminer le résultat 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 mutuellement indépendantes, la couverture MC/DC pour une décision peut généralement être atteinte en exerçant N+1 fois la décision. Le test des Conditions/Décisions modifiées nécessite des paires de test qui montre que le changement d'une seule condition atomique peut affecter indépendamment le résultat d'une décision. Notez qu'un seul cas de test peut exercer plusieurs combinaisons de conditions et qu'il n'est donc pas toujours nécessaire d'exécuter N+1 des cas de test distincts pour atteindre la couverture MC/DC.

Applicabilité
Cette technique est utilisée dans l'industrie aérospatiale et automobile et d'autres secteurs industriels pour les systèmes à sécurité critique. Elle est utilisée lors du test d'un logiciel dans lequel une défaillance pourrait provoquer une catastrophe. Le test des conditions/décisions modifiées peuvent constituer un juste milieu raisonnable entre le test des décisions et le test des conditions multiples (en raison du grand nombre de combinaisons à tester). Il est plus rigoureux que le test des décisions, mais nécessite beaucoup moins de conditions de test à exercer que le test des conditions multiples lorsqu'il y a plusieurs conditions atomiques dans la décision.

Limitations/Difficulties
La réalisation de la couverture des Conditions/Décisions Modifiées peut être compliquée lorsqu'il y a plusieurs occurrences d'une même 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 en utilisant le test des Conditions/Décisions Modifiées. L'autre approche consiste à analyser chaque décision dans laquelle un couplage existe.

Certains compilateurs et/ou interpréteurs sont conçus de telle sorte qu'ils mettent en évidence des comportements 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, pour 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. Habituellement, il est possible de configurer le compilateur pour désactiver l'optimisation de court-circuit pour les tests, mais cela peut ne pas être autorisé pour les applications critiques pour la sécurité, où le code testé et le code livré doivent être identiques.

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 de tests est appelé test des conditions multiples. En supposant N conditions atomiques uniques et mutuellement indépendantes, une couverture complète des conditions multiples pour une décision peut être obtenue en l'exerçant 2^N fois. Notez qu'un seul cas de test peut exercer plusieurs combinaisons de conditions et qu'il n'est donc pas toujours nécessaire d'exécuter 2^N cas de test distincts pour obtenir une couverture des conditions multiples de 100%.

La couverture est mesurée comme le nombre de combinaisons de conditions atomiques exercées sur l'ensemble des décisions de l'objet de test, normalement exprimée en pourcentage.

Applicabilité
Cette technique est utilisée pour tester des logiciels à haut risque et des logiciels embarqués qui sont supposés fonctionner de manière fiable sans défaillance pendant de longues périodes de temps.

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 pour le test des conditions multiples rend le test des Conditions/Décisions Modifiées plus réalisable lorsqu'il y a plusieurs conditions atomiques dans une décision.

Si le compilateur utilise le court-circuit, le nombre de combinaisons de conditions qui peut être exercé sera souvent réduit, selon l'ordre et le regroupement des opérations logiques qui sont réalisées sur les conditions atomiques.

2.6 Test des Chemins

Ce chapitre a été supprimé dans la version v4.0 de ce syllabus.

2.7 Test d'API

Une interface de programmation d'application (ou API pour Application Programming Interface) est une interface définie qui permet à un programme d'appeler un autre système logiciel, qui lui fournit un service, tel que l'accès à une ressource distante. Les services typiques incluent les web services, les bus de services d'entreprise, les bases de données, les mainframes et les interfaces utilisateur Web.

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 pour 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 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 d'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 toutes les fonctions possibles sont appelées au moins une fois.

Representational State Transfer (REST) est un type d'architecture. Les web services RESTful permettent aux systèmes demandeurs d'accéder aux ressources Web en utilisant un ensemble uniforme d'opérations sans état. Plusieurs critères de couverture existent pour les API Web RESTful, le standard de facto pour l'intégration logicielle [Web-7]. Ils peuvent être divisés en deux groupes : les critères de couverture des entrées et les critères de couverture des sorties. Entre autres, les critères d'entrée peuvent nécessiter l'exécution de toutes les opérations d'API possibles, l'utilisation de tous les paramètres d'API possibles et la couverture de séquences d'opérations d'API. Entre autres, les critères de sortie peuvent exiger la génération de tous les codes d'état corrects et erronés, et la génération de réponses contenant des ressources présentant toutes les propriétés (ou tous les types de propriétés).

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, la duplication de transactions et des problèmes dans la gestion des exceptions.

2.8 Sélectionner une Technique de Test Boîte Blanche

La technique de test boîte blanche sélectionnée est normalement spécifiée en termes de niveau de couverture requis, qui est obtenu en appliquant la technique de test. Par exemple, l'exigence d'atteindre une couverture de 100 % des instructions conduirait généralement à l'utilisation du test des instructions. Cependant, les techniques de test de boîte noire sont normalement appliquées en premier, la couverture est ensuite mesurée et la technique de test boîte blanche n'est utilisée que si le niveau de couverture boîte blanche requis n'a pas été atteint. Dans certaines situations, les tests boîte blanche peuvent être utilisés de manière moins formelle pour fournir une indication de l'endroit où la couverture peut devoir être augmentée (par exemple, la création de tests supplémentaires où les niveaux de couverture boîte blanche sont particulièrement faibles). Le test des instructions serait normalement suffisant pour une telle mesure informelle de la couverture.

Lorsque vous spécifiez un niveau de couverture boîte blanche requis, la bonne pratique est de ne le spécifier qu'à 100%. La raison en est que si des niveaux de couverture plus faibles sont requis, cela signifie généralement que les parties du code qui ne sont pas exercées par les tests sont les parties les plus difficiles à tester, et ces parties sont normalement aussi les plus complexes et les plus sujettes aux

erreurs. Ainsi, en demandant et en atteignant par exemple une couverture de 80%, cela peut signifier que le code qui inclut la majorité des défauts détectables n'est pas testé. Pour cette raison, lorsque les critères de couverture boîte blanche sont spécifiés dans les normes, ils sont presque toujours spécifiés à 100%. Des définitions strictes des niveaux de couverture ont parfois rendu ce niveau de couverture impraticable. Cependant, ceux donnés dans l'ISO 29119-4 permettent d'écarter les éléments de couverture irréalisables des calculs, faisant ainsi d'une couverture de 100% un objectif réalisable.

Lors de la spécification de la couverture boîte blanche requise pour un objet de test, il est également nécessaire de la spécifier uniquement pour un seul critère de couverture (par exemple, il n'est pas nécessaire d'exiger à la fois une couverture des instructions de 100% et une couverture MC/DC de 100%). Avec des critères de sortie à 100%, il est possible de relier certains critères de sortie dans une hiérarchie inclusive, où les critères de couverture sont montrés pour englober d'autres critères de couverture. On dit qu'un critère de couverture en englobe un autre si, pour tous les composants et leurs spécifications, chaque ensemble de cas de test qui satisfait au premier critère satisfait également au second. Par exemple, la couverture des branches englobe la couverture des instructions parce que si la couverture des branches est atteinte (à 100 %), alors la couverture des instructions à 100 % est garantie. Pour les techniques de test boîte blanche couvertes dans ce syllabus, nous pouvons dire que la couverture des branches et des décisions englobe la couverture des instructions, que le MC/DC englobe la couverture des décisions et des branches, et que la couverture des conditions multiples englobe MC/DC (si nous considérons que la couverture des branches et des décisions est la même à 100%, alors nous pouvons dire qu'elles s'englobent mutuellement).

Notez que lors de la détermination des niveaux de couverture boîte blanche à atteindre pour un système, il est tout à fait normal de définir différents niveaux pour différentes parties du système. En effet, différentes parties d'un système contribuent différemment au risque. Par exemple, dans un système avionique, les sous-systèmes associés au divertissement en vol se verraient attribuer un niveau de risque inférieur à celui associé aux commandes de vol. Tester les interfaces est courant pour tous les types de systèmes et est normalement requis pour tous les niveaux d'intégrité des systèmes liés à la sécurité (voir la section 2.8.2 pour plus d'informations sur les niveaux d'intégrité). Le niveau de couverture requis pour les tests d'API augmentera normalement en fonction du risque associé (par exemple, le niveau de risque plus élevé associé à une interface publique peut nécessiter des tests d'API plus rigoureux).

Le choix de la technique de test boîte blanche à utiliser est généralement basé sur la nature de l'objet de test et ses risques perçus. Si l'objet de test est considéré comme étant lié à la sécurité (c.-à-d. qu'une défaillance pourrait causer des dommages aux personnes ou à l'environnement), alors les normes réglementaires sont applicables et définiront les niveaux de couverture boîte blanche requis (voir la section 2.8.2). Si l'objet de test n'est pas lié à la sécurité, le choix des niveaux de couverture boîte blanche à atteindre est plus subjectif, mais devrait tout de même être largement basé sur les risques perçus, comme décrit à la section 2.8.1.

2.8.1 Systèmes non liés à la sécurité

Les facteurs suivants (sans ordre de priorité particulier) sont généralement pris en compte lors de la sélection des techniques de test boîte blanche pour les systèmes non liés à la sécurité :

- Contrat – Si le contrat exige qu'un niveau particulier de couverture soit atteint, le fait de ne pas atteindre ce niveau de couverture entraînera potentiellement une rupture de contrat.
- Client – Si le client demande un niveau de couverture particulier, par exemple dans le cadre de la planification du test, le fait de ne pas atteindre ce niveau de couverture peut entraîner des problèmes avec le client.
- Norme réglementaire – Pour certains secteurs de l'industrie (p. ex., la finance), une norme réglementaire qui définit les critères de couverture boîte blanche requis s'applique aux

systèmes essentiels à la mission. Voir la section 2.8.2 pour la couverture des normes réglementaires pour les systèmes liés à la sécurité.

- Stratégie de test – Si la stratégie de test de l'organisation spécifie les exigences pour la couverture boîte blanche du code, le fait de ne pas s'aligner sur la stratégie organisationnelle peut entrainer une censure par la haute direction.
- Style de codage – Si le code est écrit sans conditions multiples dans les décisions, ce serait une perte de temps d'exiger des niveaux de couverture boîte blanche tels que MC/DC et couverture des conditions multiples.
- Données historiques sur les défauts – Si les données historiques sur l'efficacité de l'atteinte d'un niveau de couverture particulier suggèrent qu'il serait approprié de l'utiliser pour cet objet de test, il serait risqué d'ignorer les données disponibles. Notez que ces données peuvent être disponibles au sein du projet, de l'organisation ou de l'industrie.
- Compétences et expérience – Si les testeurs disponibles pour effectuer les tests ne sont pas suffisamment expérimentés et compétents dans une technique boîte blanche particulière, cela peut être mal compris et peut introduire un risque inutile si cette technique a été sélectionnée.
- Outils – La couverture boîte blanche ne peut être mesurée dans la pratique qu'à l'aide d'outils de couverture. S'il n'existe pas de tels outils qui soutiennent une mesure de couverture donnée, le choix de cette mesure comme objectif introduirait un niveau de risque élevé.

Lorsqu'il sélectionne des tests boîte blanche pour des systèmes non liés à la sécurité, l'analyste technique de test a plus de liberté pour recommander la couverture boîte blanche appropriée pour les systèmes non liés à la sécurité que pour les systèmes liés à la sécurité. De tels choix sont généralement un compromis entre les risques perçus et le coût, les ressources et le temps nécessaires pour traiter ces risques par le biais de tests boîte blanche. Dans certaines situations, d'autres solutions, qui pourraient être mis en œuvre par d'autres approches de test logiciels ou tout autre choses (par exemple, différentes approches de développement) peuvent être plus appropriés.

2.8.2 Systèmes liés à la sécurité

Lorsque le logiciel testé fait partie d'un système lié à la sécurité, une norme réglementaire qui définit les niveaux de couverture requis à atteindre devra normalement être utilisée. Ces normes exigent généralement qu'une analyse des dangers soit effectuée pour le système et les risques qui en résultent sont utilisés pour attribuer des niveaux d'intégrité à différentes parties du système. Les niveaux de couverture requis sont définis pour chacun des niveaux d'intégrité.

IEC 61508 (sécurité fonctionnelle des systèmes programmables, électroniques et liés à la sécurité [IEC 61508]) est une norme internationale générique utilisée à ces fins. En théorie, elle pourrait être utilisée pour n'importe quel système lié à la sécurité, mais certaines industries ont créé des variantes spécifiques (par exemple, ISO 26262 [ISO 26262] s'applique aux systèmes automobiles) et certaines industries ont créé leurs propres normes (par exemple, DO-178C [DO-178C] pour les logiciels aéroportés). Des informations supplémentaires concernant la norme ISO 26262 sont fournies dans le syllabus ISTQB® Automotive Software Tester [CTSL_AuT_SYL].

IEC 61508 définit quatre niveaux d'intégrité de sécurité (SILs pour Safety Integrity Level), chacun étant défini comme un niveau relatif de réduction des risques fourni par une fonction de sécurité, corrélé à la fréquence et à la gravité des dangers perçus. Lorsqu'un objet de test remplit une fonction liée à la sécurité, plus le risque de défaillance est élevé, plus l'objet de test doit avoir une grande fiabilité. Le tableau suivant indique les niveaux de fiabilité associés aux SILs. Notez que le niveau de fiabilité pour un SIL 4 pour le cas d'un fonctionnement continu est extrêmement élevé, car il correspond à un temps moyen entre les défaillances (MTBF) supérieur à 10 000 ans.

IEC 61508 SIL	Fonctionnement continu (probabilité d'une défaillance dangereuse par heure)	Sur demande (probabilité de défaillance sur demande)
1	$\geq 10^{-6}$ to $<10^{-5}$	$\geq 10^{-2}$ to $<10^{-1}$
2	$\geq 10^{-7}$ to $<10^{-6}$	$\geq 10^{-3}$ to $<10^{-2}$
3	$\geq 10^{-8}$ to $<10^{-7}$	$\geq 10^{-4}$ to $<10^{-3}$
4	$\geq 10^{-9}$ to $<10^{-8}$	$\geq 10^{-5}$ to $<10^{-4}$

Les recommandations pour les niveaux de couverture boîte blanche associés à chaque SIL sont présentées dans le tableau suivant. Lorsqu'une entrée est indiquée comme « fortement recommandée », dans la pratique, l'atteinte de ce niveau de couverture est normalement considérée comme obligatoire. En revanche, lorsqu'une entrée n'est indiquée que comme « recommandée », de nombreux praticiens considèrent que cela est facultatif et évitent de l'atteindre en fournissant une justification appropriée. Ainsi, un objet de test affecté à un SIL 3 est normalement testé pour atteindre une couverture de branche de 100% (il atteint automatiquement une couverture de 100% des instructions, comme le montre l'ordre des hiérarchie).

IEC 61508 SIL	100% de couverture des instructions	100% de couverture des branches (décisions)	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**

Notez que les SIL ci-dessus et les exigences sur les niveaux de couverture de IEC 61508 sont différents dans ISO 26262, et différents également dans DO-178C.

3. Analyse Statique et Dynamique - 180 mins.

Mots clés

anomalie, analyse du flot de contrôle, complexité cyclomatique, analyse du flot de données, paire définition-utilisation, analyse dynamique, fuite mémoire, analyse statique, pointeur sauvage

Objectifs d'apprentissage pour Analyse Statique et Dynamique

3.2 Analyse statique

TTA-3.2.1 (K3) Utiliser l'analyse du flot de contrôle pour détecter si le code a des anomalies de flot de contrôle et pour mesurer la complexité cyclomatique

TTA-3.2.2 (K3) Utiliser l'analyse de flot de données 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

Note : TTA-3.2.4 a été supprimé dans la version v4.0 de ce syllabus

3.3 Analyse dynamique

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

3.1 Introduction

L'analyse statique (section 3.2) est une forme de test qui est réalisé sans exécuter le logiciel. La qualité du logiciel est évaluée soit par un outil, soit par une personne sur la base de sa forme, de sa structure, de son contenu ou de sa documentation. Cette vue statique du logiciel permet une analyse détaillée sans avoir à créer les données et les préconditions nécessaires pour exécuter des cas de test.

Outre l'analyse statique, les techniques de test statique comprennent également différentes formes de revues. Celles qui sont pertinentes pour l'Analyste Technique de Test sont couvertes au chapitre 5.

L'analyse dynamique (voir 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 la surveillance par une personne 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 grâce à l'utilisation d'un graphe de flot de contrôle, généralement en utilisant 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., avec plusieurs points d'entrée, ou qui ne se terminent pas), des cibles ambiguës d'appels de fonctions dans certains langages, un séquençage incorrect des opérations, du code inaccessible, des fonctions non appelées, etc.

L'analyse du flot de contrôle peut être utilisée pour déterminer la complexité cyclomatique. La complexité cyclomatique est un entier positif qui représente le nombre de chemins indépendants dans un graphe fortement connecté.

La complexité cyclomatique est généralement utilisée comme un indicateur de la complexité d'un composant. 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 ont mis en évidence cette corrélation entre la complexité et le nombre de défauts. Tout composant qui est mesuré avec une plus grande complexité doit être revu pour un possible refactoring, par exemple une division en plusieurs composants.

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 le long d'un chemin de flot de contrôle est étudié, (c.-à-d., où elle est déclarée, définie, utilisée et détruite), puisque des anomalies potentielles peuvent être identifiées si ces opérations ne sont pas utilisées dans le bon ordre [Beizer90].

Une technique courante classifie l'utilisation d'une variable en trois actions atomiques :

- lorsque la variable est définie, déclarée ou initialisée (par exemple, x:=3)
- lorsque la variable est utilisée ou lue (par exemple, if x > temp)

- lorsque la variable est tuée, détruite ou sort de son champ d'application (p. ex., text_file_1.close, variable de contrôle de boucle (i) à la sortie de la boucle)

Les séquences de telles actions qui indiquent des anomalies potentielles incluent :

- définition suivie d'une autre définition ou variable tuée sans utilisation intermédiaire
- définition sans destruction ultérieure (p. ex., entraînant une fuite de mémoire possible pour les variables allouées dynamiquement)
- utiliser ou tuer avant la définition
- utiliser ou tuer après une destruction

Selon le langage de programmation, certaines de ces anomalies peuvent être identifiées par le compilateur, mais un outil d'analyse statique distinct peut être nécessaire pour identifier les anomalies de flot de données. Par exemple, la redéfinition sans utilisation intermédiaire est autorisée dans la plupart des langages de programmation et peut être délibérément programmée, mais elle serait signalée par un outil d'analyse du flot de données comme étant une anomalie possible qui devrait être vérifiée.

L'utilisation de chemins de flot de contrôle pour déterminer la séquence d'actions d'une variable peut conduire à la déclaration d'anomalies potentielles qui ne peuvent pas se produire dans la pratique. Par exemple, les outils d'analyse statique ne peuvent pas toujours identifier si un chemin de flot de contrôle est réalisable, car certains chemins ne sont déterminés qu'en fonction des valeurs attribuées aux variables au moment de l'exécution. Il existe également une classe de problèmes d'analyse de flot de données difficiles à identifier pour les outils, lorsque les données analysées font partie de structures de données avec des variables affectées dynamiquement, telles que des enregistrements et des tableaux. Les outils d'analyse statique ont également du mal à identifier les anomalies potentielles du flot de données lorsque les variables sont partagées entre des threads de contrôle simultanés dans un programme, car la séquence d'actions sur les données devient difficile à prévoir.

Contrairement à l'analyse du flot de données, qui est un test statique, le test du flot de données est un test dynamique dans lequel des cas de test sont générés pour exercer des « paires définition-utilisation » dans le code du programme. Le test de flot de données utilise certains des mêmes concepts que l'analyse de flot de données, car ces paires définition-utilisation sont des chemins de flot de contrôle entre une définition et une utilisation ultérieure d'une variable dans un programme.

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 analyser les défauts dans le code, et la modification du code pour corriger un défaut ou ajouter une nouvelle fonctionnalité est susceptible d'entraîner l'introduction d'autres défau.

L'analyse statique est utilisée pour vérifier la conformité aux normes et recommandations de codage ; là où du code non conforme est identifié, il peut être mis à jour pour améliorer sa maintenabilité. Ces normes et lignes directrices décrivent les pratiques de codage et de conception requises telles que les conventions de nommage, les commentaires, l'indentation et la modularisation du code. Notez que les outils d'analyse statique remontent généralement des avertissements plutôt que de détecter des défauts. Ces avertissements (par exemple sur le niveau de complexité) peuvent être émis même pour du code syntaxiquement correct.

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

- Ils recherchent du code répété. Ces sections de code peuvent être des candidats à la refactorisation en composants (bien que la surcharge à l'exécution imposé par les appels de composants puisse être un problème pour les systems temps réel).
- Ils génèrent des mesures qui sont de précieux indicateurs de la modularisation du code. Il s'agit notamment de mesures de couplage et de cohésion. Un système qui a une bonne maintenabilité aura probablement une faible mesure de couplage (le degré selon lequel les composants s'appuient les uns sur les autres pendant l'exécution) et une grande cohésion (le degré selon lequel un composant 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.

La 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

En plus de l'évaluation de la maintenabilité, les 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 dans la section 4.3 et dans le syllabus Test de la Sécurité [CTSL_SEC_SYL].

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 l'apparition de 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 la performance du système en utilisant des profileurs de code afin de fournir des informations sur le comportement du système durant l'exécution, informations 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

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 et analytiques nécessaires.

3.3.2 Détection des fuites de mémoire

Une fuite de mémoire se produit lorsque des zones de la mémoire (RAM) sont allouées à un programme, mais ne sont pas libérées par la suite lorsqu'elles ne sont plus nécessaires. Cette zone de mémoire 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 libérée 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 devrait être libérée par le « ramasse miettes » automatique.

Les fuites mémoire causent généralement des problèmes qui se développent au fil du temps - lorsqu'une quantité importante de mémoire a fuité et est devenue indisponible. Lorsque le logiciel est nouvellement installé ou lorsque le système est redémarré, la mémoire est réaffectée et les fuites de mémoire ne sont donc pas perceptibles ; les tests sont un exemple où l'allocation fréquente de mémoire peut empêcher la détection de fuites de mémoire. 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 pour certains systèmes.

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.

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. 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 d'exploitation 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 et localiser les défauts associés. Grâce à l'analyse dynamique des performances du programme, les outils aident à identifier les goulets d'étranglement pour l'efficacité de la performance et génèrent un large éventail de mesures de performance pouvant être utilisées par le développeur pour adapter les performances du système. Par exemple, des informations peuvent être fournies sur le nombre de fois qu'un composant est appelé pendant l'exécution. Les composants fréquemment appelés seront probablement des candidats à l'amélioration de la performance. Souvent, la règle de Pareto s'applique ici : un programme passe une partie disproportionnée (80%) de son temps d'exécution dans un petit nombre (20%) des composants [Andrist20].

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 seul 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 [CTSL_PT_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, 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 le test non fonctionnel doit généralement être appliqué
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 co-existence 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 et des tests système, 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.

Les tests dynamiques et les tests statiques, y compris les revues (voir chapitres 2, 3 et 5) peuvent être appliqués pour tester les caractéristiques 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 la norme, désormais abandonnée, ISO 9126 (tel qu'utilisée dans la version 2012 de ce syllabus) avec ceux de la norme plus récente 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 préparation et l'exécution des tests.

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.

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 métrique.

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 le test de la performance et le test de la 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'un projet 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
- Exigences en matière d'environnement de test
- Acquisition des outils nécessaires et formations associées
- Considérations organisationnelles
- Considérations liées à la sécurité des données

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 collecte 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 la collecte des exigences, consultez le Syllabus Avancé Test Manager [ISTQB_ALTM_SYL].

Dans les développements logiciels 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 Exigences en matière d'environnement de test

De nombreux tests techniques (p. ex., tests de sécurité, tests de fiabilité, 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 options 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

Les phases d'exécutions des tests doivent être planifiées avec soin et il est fort probable que ces tests ne puissent être exécutés qu'à des moments spécifiques (par exemple, à des périodes de faible utilisation).

4.2.3 Acquisition des outils nécessaires et formations associées

Les outils font partie de l'environnement de test. Les outils commerciaux ou les simulateurs sont particulièrement pertinents pour les tests d'efficacité de la performance 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.

4.2.4 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 composants sont répartis sur différents sites et organisations, 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 les organisations peuvent n'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 composants 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 Sécurité des données et protection des données

Les mesures de sécurité spécifiques mises en œuvre pour un système devraient être considérées 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 lois 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.

4.3 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 imprévus 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 l'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 de la performance 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 [IEC 62443-3-2] qui s'applique aux systèmes industriels d'automatisation et de contrôle.

Le Syllabus Test de Sécurité [CTSL_SEC_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 la 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
- Intégrité
- Non-répudiation
- Responsabilité
- Authenticité

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.

La section 3.2 (analyse statique) et le syllabus Tests de Sécurité [CTSL_SEC_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
- Disponibilité
- Tolérance aux fautes
- Récupération

Les tests de fiabilité concernent la capacité d'un système ou d'un logiciel à exécuter des fonctions spécifiées dans des conditions spécifiées pendant une période de temps spécifiée.

4.4.2 Test de la maturité

La maturité est le degré selon lequel le système (ou le logiciel) répond aux exigences de fiabilité dans des conditions de fonctionnement normales, qui sont généralement spécifiées à l'aide d'un profil opérationnel (voir section 4.9). Les mesures de maturité, lorsqu'elles sont utilisées, constituent souvent l'un des critères de mise en production d'un système.

Traditionnellement, la maturité a été spécifiée et mesurée pour les systèmes à haute fiabilité, tels que ceux associés à des fonctions critiques pour la sécurité (par exemple, un système de contrôle de vol d'aéronef), où l'objectif de maturité est défini dans le cadre d'une norme réglementaire. Une exigence de maturité pour un système de fiabilité aussi élevée peut être un temps moyen entre les pannes (MTBF) allant jusqu'à 10^9 heures (bien que cela soit pratiquement impossible à mesurer).

L'approche habituelle pour tester la maturité des systèmes à haute fiabilité est connue sous le nom de modélisation de la croissance de la fiabilité, et elle a normalement lieu à la fin des tests système, une fois que les tests pour d'autres caractéristiques de qualité ont été terminés et que tous les défauts associés aux défaillances détectées ont été corrigés. Il s'agit d'une approche statistique généralement

réalisée dans un environnement de test aussi proche que possible de l'environnement opérationnel. Pour mesurer un MTBF spécifié, des entrées de test sont générées en fonction du profil opérationnel, le système est exécuté et les défaillances sont enregistrées (puis corrigées). Une réduction de la fréquence de défaillance permet de prédire le MTBF à l'aide d'un modèle de croissance de la fiabilité. Lorsque la maturité est utilisée comme objectif pour des systèmes à faible fiabilité (p. ex., non liés à la sécurité), le nombre de défaillances observées au cours d'une période définie d'utilisation opérationnelle prévue (p. ex., pas plus de 2 défaillances à fort impact par semaine) peut être utilisé et peut être enregistré dans le cadre de l'accord de niveau de service (SLA) pour le système.

4.4.3 Test de la disponibilité

La disponibilité est généralement spécifiée en termes de durée pendant laquelle un système (ou un logiciel) est disponible pour les utilisateurs et d'autres systèmes dans des conditions de fonctionnement normales. Les systèmes peuvent avoir une faible maturité, mais avoir tout de même une haute disponibilité. Par exemple, un réseau téléphonique peut ne pas connecter plusieurs appels (et donc avoir une faible maturité), mais tant que le système récupère rapidement et permet les prochaines tentatives de connexion, la plupart des utilisateurs seront satisfaits. Cependant, une seule panne qui a causé une panne de réseau téléphonique pendant plusieurs heures représenterait un niveau de disponibilité inacceptable. La disponibilité est souvent spécifiée dans le cadre d'un SLA et mesurée pour les systèmes opérationnels, tels que les sites Web et les applications SaaS (Software as a Service). La disponibilité d'un système peut être décrite comme 99,999% (« cinq neufs »), auquel cas il ne devrait pas être indisponible plus de 5 minutes par an. Alternativement la disponibilité du système peut être spécifiée en termes d'indisponibilité (par exemple, le système ne doit pas être en panne pendant plus de 60 minutes par mois).

La mesure de la disponibilité avant l'exploitation (p. ex., dans le cadre de la prise de décision de mise en production) est souvent effectuée à l'aide des mêmes tests que ceux utilisés pour mesurer la maturité ; les tests sont fondés sur un profil opérationnel d'utilisation prévu sur une période prolongée et effectués dans un environnement de test aussi proche que possible de l'environnement opérationnel. La disponibilité peut être mesurée en MTTF / (MTTF + MTTR), où MTTF est le temps moyen entre les défaillances et MTTR est le temps moyen de réparation (MTTR), qui est souvent mesuré dans le cadre des tests de maintenabilité. Lorsqu'un système est de haute fiabilité et intègre la capacité de récupération (voir la section 4.4.5), nous pouvons remplacer le temps moyen de récupération par le MTTR dans l'équation lorsque le système met un certain temps à se remettre d'une défaillance.

4.4.4 Test de la tolérance aux fautes

Les systèmes (ou logiciels) avec des exigences de fiabilité extrêmement élevées intègrent souvent une conception tolérante aux fautes, permettant idéalement au système de continuer à fonctionner sans temps d'arrêt notable en cas de panne. La principale mesure de la tolérance aux pannes pour un système est la capacité du système à tolérer les défaillances. Les tests de tolérance aux pannes impliquent donc de simuler des défaillances pour déterminer si le système peut continuer à fonctionner lorsqu'une telle défaillance se produit. L'identification des conditions de défaillance potentielles à tester est une partie importante du test de la tolérance aux fautes.

Une conception tolérante aux fautes implique généralement un ou plusieurs sous-systèmes en double, offrant ainsi un niveau de redondance en cas de défaillance. Dans le cas des logiciels, ces systèmes en double doivent être développés indépendamment, afin d'éviter des modes de défaillances communs ; cette approche est connue sous le nom de programmation en version N. Les systèmes de contrôle de vol des aéronefs peuvent comporter trois ou quatre niveaux de redondance, les fonctions les plus critiques pouvant être mises en œuvre dans plusieurs variantes. Lorsque la fiabilité matérielle est un problème, un système embarqué peut fonctionner sur plusieurs processeurs différents, tandis qu'un site Web critique peut s'exécuter avec un serveur miroir (basculement ou failover) exécutant les mêmes fonctions que celles qui sont toujours disponibles pour prendre le relais en cas de défaillance

du serveur principal. Quelle que soit l'approche de tolérance aux fautes mise en œuvre, son test nécessite généralement à la fois la détection de la défaillance et la réponse ultérieure à la défaillance à tester.

Les tests d'injection de défauts testent la robustesse d'un système en présence de défauts dans l'environnement du système (par exemple, une alimentation défectueuse, des messages d'entrée mal formatés, un processus ou un service non disponible, un fichier introuvable ou une mémoire non disponible) et des défauts dans le système lui-même (par exemple, un bit inversé causé par le rayonnement cosmique, une mauvaise conception, ou un mauvais codage). Le test d'injection de défauts est une forme de test négatif - nous injectons délibérément des défauts dans le système pour nous assurer qu'il réagit de la manière attendue (c'est-à-dire en toute sécurité pour un système lié à la sécurité). Parfois, les scénarios de défauts que nous testons ne devraient jamais se produire (par exemple, une tâche logicielle ne devrait jamais « mourir » ou rester coincée dans une boucle infinie) et ne peuvent pas être simulés par des tests système traditionnels, mais avec les tests d'injection de défauts, nous créons le scénario de défaut et mesurons le comportement ultérieur du système pour nous assurer qu'il détecte et gère la défaillance.

4.4.5 Test de la récupération

La récupération est une mesure de la capacité d'un système (ou d'un logiciel) à récupérer après une défaillance, que ce soit en termes de temps nécessaire à la récupération (qui peut être à un état de fonctionnement diminué) ou de quantité de données perdues. Les approches des tests de récupération comprennent les tests de basculement et les tests de sauvegarde et de restauration; les deux incluent généralement des procédures de test basées sur des tests pratiques, occasionnels et, idéalement, non annoncés, dans des environnements opérationnels.

Les tests de sauvegarde et de restauration se concentrent sur le test des procédures en place pour minimiser les effets d'une défaillance sur les données système. Les tests évaluent les procédures de sauvegarde et de restauration des données. Bien que les tests de sauvegarde des données soient relativement faciles, les tests de restauration d'un système à partir de données sauvegardées peuvent être plus complexes et nécessitent souvent une planification minutieuse pour s'assurer que les perturbations du système opérationnel sont minimisées. Les mesures comprennent le temps nécessaire pour effectuer différents types de sauvegarde (par exemple, complète et incrémentielle), le temps nécessaire pour restaurer les données (objectif de temps de récupération) et le niveau de perte de données acceptable (objectif de point de récupération).

Le test de basculement est effectué lorsque l'architecture système comprend à la fois un système principal et un système de basculement qui prendra le relais en cas de défaillance du système principal. Lorsqu'un système doit être en mesure de se remettre d'une défaillance catastrophique (par exemple, une inondation, une attaque terroriste ou une attaque de ransomware grave), les tests de basculement sont souvent appelés tests de reprise après sinistre et le ou les systèmes de basculement peuvent souvent se trouver dans un autre emplacement géographique. L'exécution d'un test complet de reprise après sinistre sur un système opérationnel nécessite une planification extrêmement minutieuse en raison des risques et des perturbations (souvent du temps libre des cadres seniors, qui est susceptible d'être consommé dans la gestion de la reprise). Si un test de reprise après sinistre complet échoue, nous reviendrons immédiatement au système principal (car il n'a pas vraiment été détruit !). Les tests de basculement incluent le test du passage au système de basculement, une fois qu'il a pris le relais, et qu'il fournit le niveau de service requis.

4.4.6 Planification des tests de fiabilité

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

- Calendrier – Les tests de fiabilité nécessitent généralement que le système complet soit testé et que d'autres types de tests soient déjà terminés – et peuvent prendre beaucoup de temps à effectuer.
- Coûts – Les systèmes à haute fiabilité sont notoirement coûteux à tester en raison des longues périodes pendant lesquelles les systèmes doivent être testés sans défaillance pour pouvoir prédire un MTBF élevé requis.
- Durée – Les tests de maturité à l'aide de modèles de croissance de la fiabilité sont basés sur les défaillances détectées et pour des niveaux de fiabilité élevés, il faudra beaucoup de temps pour obtenir des résultats statistiquement significatifs.
- Environnement de test – L'environnement de test doit être aussi similaire que possible à la production, sinon l'environnement opérationnel peut être utilisé. Cependant, si vous utilisez l'environnement opérationnel, cela peut perturber les utilisateurs et peut être à haut risque si, par exemple, un test de reprise après sinistre affecte négativement le système opérationnel.
- Portée – Différents sous-systèmes et composants peuvent être testés pour différents types et niveaux de fiabilité.
- Critères de sortie – Les exigences de fiabilité devraient être établies par des normes réglementaires pour les applications liées à la sécurité.
- Défaillance – Les mesures de fiabilité dépendent beaucoup du comptage des défaillances et il doit donc y avoir un accord préalable sur ce qui constitue une défaillance.
- Développeurs – Pour les tests de maturité utilisant des modèles de croissance de fiabilité, un accord doit être conclu avec les développeurs afin que les défauts identifiés soient corrigés dès que possible.
- La mesure de la fiabilité opérationnelle est relativement simple par rapport à la mesure de la fiabilité avant la mise en production, car nous ne devons mesurer que les défaillances; cela peut nécessiter une liaison avec le personnel opérationnel.
- Tests précoces - Pour atteindre une fiabilité élevée (par opposition à la mesure de la fiabilité), les tests nécessitent de commencer le plus tôt possible, avec des revues rigoureuses des premiers documents de référence et l'analyse statique du code.

4.4.7 Spécification des tests de fiabilité

Pour tester la maturité et la disponibilité, les tests sont largement basés sur le test du système dans des conditions de fonctionnement normales. Pour de tels tests, un profil opérationnel qui définit la manière dont le système est censé être utilisé est requis. Voir la section 4.9 pour plus de détails sur les profils opérationnels.

Pour tester la tolérance aux fautes et la capacité de récupération, il est souvent nécessaire de générer des tests qui répliquent les défaillances de l'environnement et du système lui-même, afin de déterminer comment le système réagit. Les tests d'injection de défauts sont souvent utilisés pour cela. Diverses techniques et listes de contrôle sont disponibles pour l'identification des défauts possibles et des défaillances correspondantes (par exemple, analyse par arbre de défaillance, analyse du mode de défaillance et de ses effets).

4.5 Tests de performance

4.5.1 Introduction

La classification ISO 25010 des caractéristiques qualité des produits définit les sous-caractéristiques suivantes de l'efficacité de la performance : comportement dans le temps, utilisation des ressources et

capacité. Les tests de performance (associés à la caractéristique qualité de l'efficacité de la performance) concernent la mesure de la performance d'un système ou d'un logiciel dans des conditions spécifiées par rapport à la quantité de ressources utilisées. Les ressources typiques incluent le temps écoulé, le temps CPU, la mémoire et la bande passante.

4.5.2 Test du comportement dans le temps

Le test du comportement dans le temps mesure les aspects suivants d'un système (ou d'un logiciel) dans des conditions de fonctionnement spécifiées :

- le temps écoulé entre la réception d'une demande et la première réponse (c.-à-d. le temps pour commencer à répondre, et non le temps pour terminer l'activité demandée), également appelé temps de réponse;
- délai d'exécution entre le début d'une activité et la fin de l'activité, également appelé délai de traitement;
- nombre d'activités effectuées par unité de temps (p. ex., nombre d'opérations de base de données par seconde), également appelé débit.

Pour de nombreux systèmes, des temps de réponse maximaux pour différentes fonctions du système sont spécifiés en tant qu'exigences. Dans de tels cas, le temps de réponse est le temps écoulé plus le temps d'exécution. Lorsqu'un système doit effectuer un certain nombre d'étapes (par exemple, un pipeline) pour terminer une activité, il peut être utile de mesurer le temps pris pour chaque étape et d'analyser les résultats pour déterminer si une ou plusieurs étapes causent un goulot d'étranglement.

4.5.3 Test de l'utilisation des ressources

Les tests d'utilisation des ressources mesurent les aspects suivants d'un système (ou d'un logiciel) dans des conditions de fonctionnement spécifiées :

- utilisation du processeur, normalement en pourcentage du temps CPU disponible;
- utilisation de la mémoire, normalement en pourcentage de la mémoire disponible;
- utilisation des périphériques d'E/S, normalement en pourcentage du temps disponible pour le périphérique d'E/S ;
- utilisation de la bande passante, normalement en pourcentage de la bande passante disponible.

4.5.4 Test de la capacité

Les tests de capacité mesurent les limites maximales pour les aspects suivants d'un système (ou d'un logiciel) dans des conditions de fonctionnement spécifiées :

- les transactions traitées par unité de temps (p. ex., maximum de 687 mots traduits par minute);
- les utilisateurs accédant simultanément au système (p. ex., un maximum de 1223 utilisateurs);
- les nouveaux utilisateurs ajoutés ayant accès au système par unité de temps (p. ex., maximum de 400 utilisateurs ajoutés par seconde).

4.5.5 Aspects communs du test de la performance

Lors du test du comportement dans le temps, de l'utilisation des ressources ou de la capacité, il est normal que plusieurs mesures soient prises et que la moyenne soit utilisée comme mesure déclarée ; en effet, les valeurs de temps mesurées peuvent fluctuer en fonction d'autres tâches en arrière-plan que le système peut effectuer. Dans certaines situations, les mesures seront traitées de manière plus méticuleuse (par exemple, en utilisant la variance ou d'autres mesures statistiques), ou les valeurs aberrantes peuvent être étudiées et écartées, le cas échéant.

L'analyse dynamique (voir la section 3.3.4) peut être utilisée pour identifier les composants à l'origine d'un goulot d'étranglement, mesurer les ressources utilisées pour les tests d'utilisation des ressources et mesurer les limites maximales pour les tests de capacité.

4.5.6 Types de tests de performance

Les tests de performance diffèrent de la plupart des autres formes de tests en ce qu'il peut y avoir deux objectifs distincts. Le premier consiste à déterminer si le logiciel testé répond aux critères d'acceptation spécifiés. Par exemple, déterminer si le système affiche une page Web demandée dans les 4 secondes maximums spécifiées. Le deuxième objectif est de fournir des informations aux développeurs du système pour les aider à améliorer l'efficacité du système. Par exemple, détecter les goulots d'étranglement et identifier les parties de l'architecture du système qui sont affectées lorsqu'un nombre étonnamment élevé d'utilisateurs accèdent simultanément au système.

Les tests de performance décrits aux sections 4.5.2, 4.5.3 et 4.5.4 peuvent tous être utilisés pour déterminer si le logiciel testé répond aux critères d'acceptation spécifiés. Ils sont également utilisés pour mesurer les valeurs de référence qui sont utilisées pour une comparaison ultérieure lorsque le système est modifié. Les types de test de performance suivants sont plus souvent utilisés pour fournir des informations aux développeurs sur la façon dont le système réagit dans différentes conditions opérationnelles.

4.5.6.1 Test de charge

Les tests de charge se concentrent sur la capacité d'un système à gérer différentes charges. Ces charges sont généralement définies en termes de nombre d'utilisateurs accédant simultanément au système ou de nombre de processus simultanés en cours d'exécution et peuvent être définies comme des profils opérationnels (voir la section 4.9 pour plus de détails sur les profils opérationnels). La gestion de ces charges est généralement mesurée en termes de comportement dans le temps du système et d'utilisation des ressources (par exemple, déterminer l'effet sur le temps de réponse du doublement du nombre d'utilisateurs). Lors de tests de charge, il est normal de commencer avec une faible charge et d'augmenter progressivement la charge tout en mesurant le comportement dans le temps du système et l'utilisation des ressources. Les informations typiques des tests de charge qui pourraient être utiles aux développeurs incluraient des modifications inattendues des temps de réponse ou de l'utilisation des ressources système lorsque le système gérait une charge particulière.

4.5.6.2 Test de stress

Il existe deux types de tests de stress ; le premier est similaire au test de charge et le second est une forme de test de robustesse.

Dans le premier, les tests de charge sont normalement effectués avec la charge initialement réglée au maximum attendu, puis augmentée jusqu'à ce que le système tombe en panne (par exemple, les temps de réponse deviennent déraisonnablement longs ou le système cesse de fonctionner). Parfois, au lieu de forcer le système à tomber en panne, une charge élevée est utilisée pour stresser le système, puis la charge est réduite à un niveau normal et le système est vérifié pour s'assurer que ses niveaux de performance ont retrouvé leurs niveaux d'avant la contrainte.

Dans le second, des tests de performances sont exécutés avec le système délibérément compromis en réduisant son accès aux ressources attendues (par exemple, en réduisant la mémoire ou la bande passante disponible). Les résultats des tests de stress peuvent fournir aux développeurs un aperçu des aspects du système qui sont les plus critiques (c'est-à-dire les maillons faibles) et peuvent donc nécessiter une mise à niveau.

4.5.6.3 Test d'évolutivité

Un système évolutif peut s'adapter à différentes charges. Par exemple, un site Web évolutif pourrait évoluer pour utiliser plus de serveurs back-end à mesure que la demande augmente et en utiliser moins

lorsque la demande diminue. Les tests d'évolutivité sont similaires aux tests de charge, mais testent la capacité d'un système à évoluer vers le haut et vers le bas lorsqu'il est confronté à des charges changeantes (par exemple, plus d'utilisateurs que le matériel actuel ne peut gérer).

Le syllabus Tests de Performance [CTSL_PT_SYL] comprend d'autres types de tests de performance.

4.5.7 Planification des tests de performance

En général, les éléments suivants sont particulièrement pertinents lors de la planification des tests d'efficacité de la performance :

- Calendrier – Les tests de performance nécessitent souvent que l'ensemble du système soit implémenté et exécuté sur un environnement de test représentatif, ce qui signifie qu'ils sont généralement effectués dans le cadre des tests système.
- Revues - Les revues de code, en particulier celles qui se concentrent sur l'interaction avec la base de données, l'interaction entre composants et la gestion des erreurs, peuvent identifier les problèmes d'efficacité de la performance (en particulier en ce qui concerne la logique « attente et nouvelle tentative » et les requêtes inefficaces) et doivent être planifiées pour avoir lieu dès que le code est disponible (c'est-à-dire avant les tests dynamiques).
- Tests précoces – Certains tests de performance (par exemple, déterminer l'utilisation du processeur pour un composant critique) peuvent être planifiés dans le cadre des tests de composants. Les composants identifiés comme étant un goulot d'étranglement par les tests de performance peuvent être mis à jour et testés à nouveau isolément dans le cadre des tests de composants.
- Modifications de l'architecture - Les résultats défavorables des tests de performance peuvent parfois entraîner une modification de l'architecture du système. Lorsque des changements aussi importants du système pourraient être suggérés par le résultat des tests de performance, les tests de performance devraient commencer le plus tôt possible, afin de maximiser le temps disponible pour résoudre ces problèmes.
- Coûts – Les outils et les environnements de test peuvent être coûteux, ce qui signifie que des environnements de test temporaires basés sur le cloud peuvent être loués et que des licences d'outils « à la demande » peuvent être utilisées. Dans de tels cas, la planification des tests doit généralement optimiser le temps passé à exécuter les tests afin de minimiser les coûts.
- Environnement de test - L'environnement de test doit être aussi représentatif que possible de l'environnement opérationnel, sinon le défi de la mise à l'échelle des résultats des tests de performance de l'environnement de test vers l'environnement opérationnel attendu est accru.
- Critères de sortie – Les exigences d'efficacité de la performance peuvent parfois être difficiles à obtenir du client et sont donc souvent dérivées des bases de référence de systèmes précédents ou similaires. Dans le cas des systèmes embarqués liés à la sécurité, certaines exigences, telles que la quantité maximale de CPU et de mémoire utilisée, peuvent être spécifiées par des normes réglementaires.
- Outils – Des outils de génération de charge sont souvent nécessaires pour soutenir les tests de performance. Par exemple, la vérification de l'évolutivité d'un site Web populaire peut nécessiter la simulation de centaines de milliers d'utilisateurs virtuels. Les outils qui simulent les restrictions de ressources sont également particulièrement utiles pour les tests de stress. Il faut veiller à ce que tous les outils acquis pour soutenir les tests soient compatibles avec les protocoles de communication utilisés par le système testé.

Le syllabus Tests de Performance [CTSL_PT_SYL] comprend plus de détails sur la planification des tests de performance.

4.5.8 Spécification des tests de performance

Les tests de performance sont largement basés sur le test du système dans des conditions de fonctionnement spécifiées. Pour de tels tests, un profil opérationnel qui définit la manière dont le système est censé être utilisé est requis. Voir la section 4.9 pour plus de détails sur les profils opérationnels.

Pour les tests de performance, il est souvent nécessaire de modifier la charge sur le système en modifiant des parties du profil opérationnel pour simuler un changement par rapport à l'utilisation opérationnelle attendue du système. Par exemple, dans le cas des tests de capacité, il sera normalement nécessaire de remplacer le profil opérationnel concernant la variable testée pour la capacité (par exemple, augmenter le nombre d'utilisateurs accédant au système jusqu'à ce que le système cesse de répondre pour déterminer la capacité d'accès de l'utilisateur). De même, le volume des transactions peut être progressivement augmenté lors des tests de charge.

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

4.6 Tests de maintenabilité

Les logiciels passent souvent une bien plus grande partie de leur vie à être maintenus qu'à être développés. 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, testé, modularisé et réutilisé.Les tests de maintenabilité ne doivent pas être confondus avec les tests de maintenance, qui sont effectués pour tester les modifications apportées au logiciel opérationnel.

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 statiques 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, 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 développées pour maintenir une application particulière (p. ex., pour effectuer des mises à niveau logicielles). Des 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 de:

- Analysabilité
- Modifiabilité
- Testabilité

Les facteurs qui influent sur ces caractéristiques 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 d'interface).

Les autres sous-caractéristiques de qualité pertinentes pour la maintenabilité [ISO 25010] sont les suivantes :

- Modularité
- Réutilisabilité

La modularité peut être testée au moyen d'une analyse statique (voir section 3.2.3). Les tests de réutilisabilité peuvent prendre la forme de revues architecturales (voir chapitre 5).

4.7 Tests de portabilité

4.7.1 Introduction

En général, les tests de portabilité sont liés au degré selon lequel un composant logiciel ou un système peut être transféré dans son environnement cible (soit pour la première fois, soit à partir d'un environnement existant), peut être adapté à un nouvel environnement, ou peut remplacer une autre entité.

ISO 25010 [ISO25010] comprend les sous-caractéristiques suivantes de la portabilité :

- Facilité d'installation
- Adaptabilité
- Facilité de remplacement

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 sur 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 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 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

Des tests d'aptitude fonctionnelle 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.). Pour spécifier les tests d'adaptabilité, il faut que les 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.

4.7.4 Test de facilité de remplacement

Les tests de facilité de remplacement se concentrent sur la capacité d'un composant logiciel à remplacer un composant logiciel existant dans un système. Cela peut être particulièrement pertinent pour les systèmes qui utilisent des composants sur étagère pour des composants spécifiques du système ou pour des applications de l'internet des objets (IoT).

Les tests de facilité de remplacement peuvent être exécutés 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 aussi être évaluée par une revue technique ou une inspection aux niveaux de l'architecture et de la conception, où l'accent est mis sur la définition claire des interfaces des composants qui pourraient être utilisés pour un remplacement.

4.8 Test de compatibilité

4.8.1 Introduction

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

- Co-existence
- Interopérabilité

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 l'aptitude fonctionnelle 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.

4.9 Profils opérationnels

Les profils opérationnels sont utilisés dans le cadre de la spécification de test pour plusieurs types de tests non fonctionnels, y compris les tests de fiabilité et les tests de performance. Ils sont particulièrement utiles lorsque l'exigence testée inclut la contrainte « dans des conditions spécifiées », car ils peuvent être utilisés pour définir ces conditions.

Le profil opérationnel définit un modèle d'utilisation du système, généralement en termes d'utilisateurs du système et d'opérations effectuées par le système. Les utilisateurs sont généralement spécifiés en termes de nombre d'utilisateurs qui utilisent le système (et à quel moment), et peut-être leur type (par exemple, utilisateur principal, utilisateur secondaire). Les différentes opérations attendues par le système sont généralement spécifiées, avec leur fréquence (et leur probabilité d'occurrence). Ces informations peuvent être obtenues en utilisant des outils de surveillance (lorsque l'application réelle ou similaire est déjà disponible) ou en prédisant l'utilisation en fonction d'algorithmes ou d'estimations fournis par l'organisation métier.

Les outils peuvent être utilisés pour générer des entrées de test en fonction du profil opérationnel, souvent en utilisant une approche qui génère les entrées de test de manière pseudo-aléatoire. Ces outils peuvent être utilisés pour créer des utilisateurs « virtuels » ou simulés en quantités qui correspondent au profil opérationnel (p. ex., pour les tests de fiabilité et de disponibilité) ou qui le

dépassent (p. ex., pour les tests de stress ou de capacité). Voir la section 6.2.3 pour plus de détails sur ces outils.

5. Revues - 165 mins.

Mots clés
revue, revue technique

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 détaillée 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 dans la fourniture 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, la vérification en ce qui concerne les aspects techniques 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 éléments 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.

5.2.1 Revues d'architecture

L'architecture logicielle se compose des concepts et propriétés fondamentales d'un système, implémentés dans ses éléments, ses relations et dans les principes de sa conception et de son évolution [ISO 42010].

Les checklists[1] utilisées pour les revues d'architecture du comportement dans le temps de sites Web 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
- Isolation des processus entre Traitement Transactionnel en ligne (Online Transactional Processing (OLTP)) et Traitement Analytique en ligne (Online Analytical Processing (OLAP))
- Réplication des données

5.2.2 Revues de Code

Les checklists pour les revues de code sont nécessairement bas niveau et sont plus utiles lorsqu'elles sont spécifiques à un language. 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 ?
    - 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

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

- 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, émulateur, injection de fautes, insertion de fautes, test dirigés par mots-clés, test basé sur des modèles (MBT pour Model Based Testing), simulateur, exécution 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 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)
- Travailler avec le Test Manager pour estimer le coût de la mise en œuvre, y compris la formation. Dans les projets de développement 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 Automatisation des Tests [CTSL_TAE_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 de développement 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 d'une interface graphique utilisateur (GUI), d'une interface applicative (API) et d'une interface ligne de commande (CLI)
- 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 Automatisation des Tests [CTSL_TAE_SYL] détaille la sélection d'une approche d'automatisation.

6.1.1.1 Automatisation via GUI, API et CLI

L'automatisation des tests ne se limite pas aux tests via l'interface graphique utilisateur. Des outils existent aussi pour aider à automatiser les tests au niveau de l'API (Application Programming Interface), via une interface en ligne de commande (CLI) 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 de développement 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 de test 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. Les mots clés peuvent être définis pour des actions de haut ou de bas niveau. Par exemple, les mots clés du processus métier peuvent inclure « Connexion », « Créer Utilisateur » et « Supprimer Utilisateur ». De tels mots clés décrivent des actions de haut niveau qui seront effectuées dans le domaine de l'application. Les actions de plus bas niveau dénotent l'interaction avec l'interface logicielle elle-même. Des mots clés comme « CliquerBouton », « SélectionnerDansListe » ou « ParcourirArborescence » peuvent être utilisées pour tester des fonctionnalités d'interface graphique qui ne s'intègrent pas parfaitement dans les mots clés des processus métier. Les mots-clés peuvent contenir des paramètres, par exemple le mot-clé « Login » peut avoir deux paramètres : user_name et mot_de_passe.

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 testé 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 de chaque test. 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 de développement 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.

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 détaillés, 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 qui créent collectivement un enregistrement. Cependant, créer un mot clé de haut niveau appelant les six mots clés n'est pas forcément l'approche la plus efficiente ?
- 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. Elle peut être plus coûteuse à mettre en place au départ, mais sera certainement globalement plus économique si le porjet dure suffisament longtemps.

Le syllabus Automatisation des Tests [CTSL_TAE_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 [CTSL_MAT_SYL]
- Test de Performance [CTSL_PT_SYL]
- Model-Based Testing [CTSL_MBT_SYL]
- Automatisation des Tests [CTSL_TAE_SYL]

6.2.1 Outils d'Insertion de Fautes

Les outils d'insertion de fautes modifient 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 insertion de fautes 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 d'Injection de Fautes

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 injectées provoquent des conditions négatives, ce qui devrait entraîner l'exécution (et le test) de la gestion des erreurs. Cette perturbation du flot d'exécution normal du code améliore également la couverture de code.

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

6.2.3 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, donnant des 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 4.9) 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 les 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 simples comparé au profil de charge et aux installations fournies par les outils commerciaux. De plus amples détails sur les outils de test de performance sont fournis dans le Syllabus Tests de Performance [CTSL_PT_SYL].

6.2.4 Outils pour le test des sites 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
- Outils de test de la performance 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 code du serveur, vérifier des fichiers orphelins (non liés) précédemment référencés par le site Web
- 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.

6.2.5 Outils de Test Basé 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 [CTSL_MBT_SYL].

6.2.6 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 composant. Par exemple, si Java a été utilisé comme langage de programmation, JUnit peut être utilisé pour automatiser les tests de composant. 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.

Certains outils d'automatisation de build permettent de déclencher automatiquement un nouveau build lorsqu'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 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.

6.2.7 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 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 utilisés en remplacement des appareils réels dans les tests, mais se limitent généralement aux tests fonctionnels initiaux et à la simulation de nombreux utilisateurs virtuels dans les tests de charge. Les simulateurs sont relativement simples (par rapport aux émulateurs) et peuvent exécuter des tests plus rapidement qu'un émulateur. 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.

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 et l'accéléromètre,. 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 les émulateurs sont souvent utilisés pour réduire le coût des environnements de test en remplaçant des appareils réels. 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 ou 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 Tests d'Application Mobile [CTSL_MAT_SYL] comprend plus de détails.

7. Références

7.1 Standards

Les normes suivantes sont mentionnées dans ces chapitres respectifs.

[DO-178C]	DO-178C - Software Considerations in Airborne Systems and Equipment Certification, RTCA, 2011.
	Chapter 2
[ISO 9126]	ISO/IEC 9126-1:2001, Software Engineering – Software Product Quality
	Chapter 4 and Appendix A.
[ISO 25010]	ISO/IEC 25010: 2011, Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models
	Chapters 1, 4 and Appendix A.
[ISO 29119]	ISO/IEC/IEEE 29119-4:2015, Software and systems engineering - Software testing - Part 4: Test techniques
	Chapter 2.
[ISO 42010]	ISO/IEC/IEEE 42010:2011, Systems and software engineering - Architecture description
	Chapter 5.
[IEC 61508]	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.
[ISO 26262]	ISO 26262-1:2018, Road vehicles — Functional safety, Parts 1 to 12.
	Chapter 2.
[IEC 62443-3-2]	IEC 62443-3-2:2020, Security for industrial automation and control systems - Part 3-2: Security risk assessment for system design
	Chapter 4.

7.2 Documents ISTQB® (versions originales en anglaise)

[ISTQB_AL_TTA_OVIEW]	ISTQB® Technical Test Analyst Advanced Level Overview v4.0
[CTSL_SEC_SYL]	Advanced Level Security Testing Syllabus, Version 2016
[CTSL_TAE_SYL]	Advanced Level Test Automation Engineer Syllabus, Version 2017
[ISTQB_FL_SYL]	Foundation Level Syllabus, Version 2018
[CTSL_PT_SYL]	Foundation Level Performance Testing Syllabus, Version 2018
[CTSL_MBT_SYL]	Foundation Level Model-Based Testing Syllabus, Version 2015
[ISTQB_ALTM_SYL]	Advanced Level Test Manager Syllabus, Version 2012
[CTSL_MAT_SYL]	Foundation Level Mobile Application Testing Syllabus, 2019
[ISTQB_GLOSSARY]	Glossary of Terms used in Software Testing, Version 3.2, 2019
[CTSL_AuT_SYL]	Foundation Level Automotive Software Tester, Version 2018

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

7.3 Livres et articles

[Andrist20] Björn Andrist and Viktor Sehr, C++ High Performance: Master the art of optimizing the functioning of your C++ code, 2nd Edition, Packt Publishing, 2020

[Beizer90] Boris Beizer, "Software Testing Techniques Second Edition", International Thomson Computer Press, 1990, ISBN 1-8503-2880-3

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

[Kaner02] Cem Kaner, James Bach, Bret Pettichord; “Lessons Learned in Software Testing”; Wiley, 2002, ISBN: 0-471-08112-4

[McCabe76] Thomas J. McCabe, "A Complexity Measure", IEEE Transactions on Software Engineering, Vol. SE-2, No. 4, December 1976. PP 308-320

[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://validator.w3.org/
[Web-7] https://dl.acm.org/doi/abs/10.1145/3340433.3342822

Chapitre 2 : [Web-7]
Chapitre 4: [Web-1] [Web-4]
Chapitre 5: [Web-2]
Chapitre 6: [Web-3] [Web-5] [Web-6]

8. Appendix A : Aperçu des caractéristiques de qualité

Le tableau suivant compare les caractéristiques de qualité décrites dans la norme désormais abandonnée ISO 9126 (tel qu'il est utilisé dans la version 2012 du syllabus Analyste Technique de Test) avec celles de la norme plus récente ISO 25010 (tel qu'utilisé dans la dernière version du syllabus). Notez que l'aptitude fonctionnelle et l'utilisabilité sont couvertes dans le syllabus Analyste de Test.

ISO/IEC 25010	ISO/IEC 9126-1	Notes
Aptitude fonctionnelle	**Fonctionnalité**	Le nouveau nom est plus précis et évite la confusion avec d'autres significations de « fonctionnalité »
Complétude fonctionnelle		Couverture des besoins déclarés
Exactitude fonctionnelle	Précision	Plus général que la précision
Adéquation fonctionnelle	Adéquation	Couverture des besoins implicites
	Interopérabilité	Déplacé vers compatibilité
	Sécurité	Maintenant une caractéristique
Efficacité de la performance	**Efficacité**	Renommé pour éviter tout conflit avec la définition de l'efficacité dans l'ISO/IEC 25062
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)
Utilisabilité		Problème de qualité implicite rendu explicite
Reconnaissance de la pertinence	Compréhensibilité	Le nouveau nom est plus précis
Apprentissage	Apprentissage	
Opérabilité		
Protection contre les erreurs humaines		Nouvelle sous-caractéristique
Esthétique de l'interface	Attractivité	Le nouveau nom est plus précis
Accessibilité		Nouvelle sous-caractéristique
Fiabilité	**Fiabilité**	
Maturité	Maturité	
Disponibilité		Nouvelle sous-caractéristique
Tolérance aux fautes	Tolérance aux fautes	
Récupération	Récupération	
Security	**Sécurité**	Pas de sous-caractéristique précédente
Confidentialité		Pas de sous-caractéristique précédente
Intégrité		Pas de sous-caractéristique précédente
Non-répudiation		Pas de sous-caractéristique précédente

Responsabilité		Pas de sous-caractéristique précédente
Authenticité		Pas de sous-caractéristique précédente
Maintenabilité	**Maintenabilité**	
Modularité		Nouvelle sous-caractéristique
Réutilisabilité		Nouvelle sous-caractéristique
Facilité d'analyse	Facilité d'analyse	
Facilité de modification	Stabilité	Nom plus précis combinant Facilité de changement et Stabilité
Testabilité	Testabilité	
Portabilité	**Portabilité**	
Adaptabilité	Adaptabilité	
Facilité d'installation	Facilité d'installation	
	Co-existence	Déplacé vers compatibilité
Facilité de remplacement	Facilité de remplacement	

9. Index