Testeur certifié niveau Avancé Automatisation des tests - Ingénierie Syllabus

Version 2.0

International Software Testing Qualifications Board

Notice de Copyright

Historique de révision

Version	Date	Remarques
Syllabus 2016	2016/10/21	CTAL-TAE soumise à l'AG
Syllabus v2.0	2024/05/03	CTAL-TAE v2.0 soumise à l'AG
Syllabus v2.0 FR	2024/07/25	CTAL-TAE v2.0 soumise à l'AG, version française

Table des matières

Notice de Copyright 2
Historique de révision 3
Table des matières 4
Remerciements 7
0 Introduction 8
0.1 Objectif de ce syllabus 8
0.2 L'Automatisation des tests – Ingénierie dans le test logiciel 8
0.3 Parcours de carrière des testeurs et des ingénieurs en Automatisation des tests 9
0.4 Objectifs métier 9
0.5 Objectifs d'apprentissage examinables et niveau cognitif de connaissance 10
0.6 L'examen d'Automatisation des tests - ingénierie 10
0.7 Accréditation 11
0.8 Gestion des normes 11
0.9 Actualisation 11
0.10 Niveau de détail 11
0.11 Organisation de ce syllabus 12
1 Introduction et objectifs de l'Automatisation des tests – 45 minutes (K-2) 14
1.1 Objectif de l'Automatisation des tests 15
1.1.1 Expliquer les avantages et les inconvénients de l'Automatisation des tests 15
1.2 L'automatisation dans le cycle de vie de développement logiciel 17
1.2.1 Expliquer comment l'Automatisation des tests est appliquée dans les différents modèles de cycles de vie de développement logiciel ? 17
1.2.2 Choisir des outils d'Automatisation des tests adaptés à un système sous test donné 17
2 Se préparer à l'Automatisation des tests – 180 minutes (K4) 19
2.1 Comprendre la configuration d'une infrastructure permettant l'Automatisation des tests 20
2.1.1 Décrire les besoins de configuration d'une infrastructure permettant l'implémentation de l'Automatisation des tests 20
2.1.2 Expliquer comment l'Automatisation des tests est exploitée au sein de différents environnements. 21
2.2 Processus d'évaluation pour sélectionner les bons outils et stratégies 22
2.2.1 Analyser un système sous test pour déterminer la solution d'Automatisation des tests appropriée 22

2.2.2 Illustrer les constatations techniques d'une évaluation d'outil 22
3 Architecture d'Automatisation des tests – 210 minutes (K3) 24
3.1 Concepts de design exploités dans l'automatisation des tests 25
3.1.1 Expliquer les principales capacités d'une architecture d'Automatisation des tests 25
3.1.2 Expliquer comment concevoir une solution d'Automatisation des tests 26
3.1.3 Appliquer une superposition dans les frameworks d'Automatisation des tests 26
3.1.4 Appliquer différentes approches pour automatiser les cas de tests 27
3.1.5 Appliquer les principes et les canevas de conception à l'Automatisation des tests 30
4 Implémentation de l'Automatisation des tests – 150 minutes (K4) 32
4.1 Développement de l'Automatisation des tests 33
4.1.1 Appliquer des lignes directrices qui soutiennent des activités efficaces de pilotage et de déploiement de l'Automatisation des tests 33
4.2 Risques associés au développement de l'Automatisation des tests 34
4.2.1 Analyse des risques de déploiement et planification des stratégies d'atténuation des risques pour l'Automatisation des tests 34
4.3 Maintenabilité de la solution d'Automatisation des tests 35
4.3.1 Expliquer quels facteurs soutiennent et affectent la maintenabilité de la solution d'Automatisation des tests 35
5 Stratégies d'implémentation et de déploiement de l'Automatisation des tests – 90 minutes (K3) 37
5.1 Intégration aux pipelines CI/CD 38
5.1.1 Appliquer l'Automatisation des tests à différents niveaux de test dans les pipelines 38
5.1.2 Expliquer la gestion de la configuration pour les testware 39
5.1.3 Expliquer les dépendances de l'Automatisation des tests pour une infrastructure d'API 40
6 Reporting et métriques sur l'Automatisation des tests – 150 minutes (K4) 41
6.1 Collecte, analyse et reporting des données de l'Automatisation des tests 42
6.1.1 Appliquer les méthodes de collecte des données de la solution d'Automatisation des tests et du système sous test 42
6.1.2 Analyser les données de la solution d'Automatisation des tests et du système sous test pour mieux comprendre les résultats de test 44
6.1.3 Expliquer comment un rapport d'avancement des tests est élaboré et publié 45
7 Vérifier la solution d'Automatisation des tests – 135 minutes (K3) 48
7.1 Vérification de l'infrastructure d'Automatisation des tests 49
7.1.1 Planifier de vérifier l'environnement d'Automatisation des tests, y compris la mise en place des outils de test 49

7.1.2 Expliquer le comportement correct pour un script de test automatisé donné et/ou une suite de tests 50

7.1.3 Identifier où l'Automatisation des tests produit des résultats inattendus 51

7.1.4 Expliquer comment l'analyse statique peut contribuer à la qualité du code d'Automatisation des tests 51

8 Amélioration continue – 210 minutes (K4) 53

8.1 Possibilités d'amélioration continue de l'Automatisation des tests 54

8.1.1 Découvrir les possibilités d'amélioration des cas de test grâce à la collecte et à l'analyse des données 54

8.1.2 Analyser les aspects techniques d'une solution d'Automatisation des tests déployée et formuler des recommandations d'amélioration. 54

8.1.3 Restructurer le logiciel de test automatisé pour l'aligner sur les mises à jour du système sous test 57

8.1.4 Résumer les possibilités d'utilisation des outils d'Automatisation des tests 59

9 Références 60

10 Appendice A – Objectifs d'apprentissage/Niveau cognitif de connaissances 63

11 Appendice B – Matrice de traçabilité des objectifs métier avec les objectifs d'apprentissage 66

12 Appendice C – Notes de version 70

13 Appendice D – Termes spécifiques au domaine 71

14 Index 72

Remerciements

Ce document a été officiellement publié par l'Assemblée Générale de l'ISTQB® le 3 mai 2024.

Il a été produit par le groupe de travail sur l'Automatisation des tests du groupe de travail Spécialiste de l'International Software Testing Qualifications Board: Graham Bath (Specialist Working Group Chair) Andrew Pollner (Specialist Working Group Vice Chair and Test Automation Task Force Chair), Péter Földházi, Patrick Quilter, Gergely Ágnecz, László Szikszai. Réviseurs du groupe de travail sur l'Automatisation des tests : Armin Beer, Armin Born, Geza Bujdoso, Renzo Cerquozzi, Jan Giesen, Arnika Hryszko, Kari Kakkonen, Gary Mogyorodi, Chris van Bael, Carsten Weise, Marc-Florian Wendland.

Revue technique : Gary Mogyorodi

Les personnes suivantes ont participé à la revue, aux commentaires et au vote de ce syllabus :

Horváth Ágota, Laura Albert, Remigiusz Bednarczyk, Jürgen Beniermann, Armin Born, Alessandro Collino, Nicola De Rosa, Wim Decoutere, Ding Guofu, Istvan Forgacs, Elizabeta Fourneret, Sudhish Garg, Jan Giesen , Matthew Gregg, Tobias Horn, Mattijs Kemmink, Hardik Kori, Jayakrishnan Krishnankutty, Ashish Kulkarni, Vincenzo Marrazzo, Marton Matyas, Patricia McQuaid, Rajeev Menon, Ingvar Nordström, Arnd Pehl, Michaël Pilaeten, Daniel Polan, Nishan Portoyan, Meile Posthuma, Adam Roman, Pavel Sharikov, Péter Sótér , Lucjan Stapp, Richard Taylor, Giancarlo Tomasig, Chris Van Bael, Koen Van Belle, Johan Van Berkel, Carsten Weise, Marc-Florian Wendland, Ester Zabar.

Groupe de travail de l'ISTQB® Automatisation des tests de niveau Avancé - Ingénierie (Edition 2016): Andrew Pollner (Chair), Bryan Bakker, Armin Born, Mark Fewster, Jani Haukinen, Raluca Popescu, Ina Schieferdecker.

0 Introduction

0.1 Objectif de ce syllabus

Ce syllabus constitue la base de la Certification Internationale en Test Logiciel pour l'Automatisation des tests - Ingénierie de niveau Avancé (CTAL-TAE). L'ISTQB® fournit ce syllabus comme suit :

1. Aux Membres de l'ISTQB®, pour qu'ils le traduisent dans leur langue locale et accréditent les organismes de formation. Les Membres peuvent adapter le syllabus à leurs besoins linguistiques particuliers et modifier les références pour les adapter à leurs publications locales.
2. Aux organismes de certification, pour qu'ils élaborent des questions d'examen dans leur langue locale, adaptées aux objectifs d'apprentissage de ce syllabus.
3. Aux organismes de formation, pour qu'ils produisent des supports de cours et déterminent les méthodes d'enseignement appropriées.
4. Aux candidats à la certification, pour qu'ils se préparent à l'examen de certification (soit dans le cadre d'un cours de formation, soit de manière indépendante).
5. A la communauté internationale de l'ingénierie des logiciels et des systèmes, pour faire progresser la profession de testeur de logiciels et de systèmes, et comme base pour des livres et des articles.

0.2 L'Automatisation des tests – Ingénierie dans le test logiciel

La certification en Automatisation des tests- Ingénierie s'adresse à toute personne impliquée dans les tests de logiciels et l'Automatisation des tests. Il s'agit notamment des testeurs, des analystes de test, des ingénieurs en Automatisation des tests, des consultants en test, des architectes de test, des Test Managers et des développeurs de logiciels. Cette certification s'adresse également à tous ceux qui souhaitent acquérir une compréhension de base de l'Automatisation des tests, tels que les chefs de projet, les responsables de la qualité, les responsables du développement de logiciels, les analystes métier, les directeurs informatiques et les consultants en management.

Le syllabus Automatisation des tests - Ingénierie s'adresse à l'ingénieur en Automatisation des tests qui cherche à implémenter ou à améliorer l'Automatisation des tests. Il définit les méthodes et les pratiques qui peuvent soutenir une solution durable.

D'autres lignes directrices et modèles de référence relatifs aux solutions d'Automatisation des tests se trouvent dans les normes de génie logiciel pour les cycles de vie du développement logiciel sélectionnés, les technologies de programmation et les normes de formatage. Ce syllabus n'enseigne pas le génie logiciel. Cependant, on attend d'un ingénieur en Automatisation des tests qu'il ait des compétences, de l'expérience et de l'expertise en ingénierie logicielle.

En outre, un ingénieur en Automatisation des tests doit connaître les normes de programmation et de documentation de l'industrie et les meilleures pratiques pour les utiliser lors du développement d'une solution d'Automatisation des tests. Ces pratiques peuvent accroître la maintenabilité, la fiabilité et la sécurité de la solution d'Automatisation des tests. Ces normes sont généralement basées sur des caractéristiques de qualité.

0.3 Parcours de carrière des testeurs et des ingénieurs en Automatisation des tests

Le programme de l'ISTQB® apporte un soutien aux professionnels du test à tous les stades de leur carrière en leur offrant des connaissances à la fois étendues et approfondies. Les personnes qui obtiennent la certification ISTQB® Automatisation des tests - Ingénierie peuvent également être intéressées par la certification Stratégie d'Automatisation des tests (CT-TAS).

Les personnes qui obtiennent la certification ISTQB® Automatisation des tests - Ingénierie peuvent également être intéressées par les principaux niveaux Avancé (Analyste de test, Analyste technique de test et Management des tests) et par la suite par le niveau Expert (Management des tests ou Amélioration du processus de test). Toute personne cherchant à développer ses compétences en matière de pratiques de test dans un environnement en mode Agile pourrait envisager les certifications Testeur technique Agile ou Conduite des tests en mode Agile à l'échelle. La filière Spécialiste offre une plongée approfondie dans les domaines qui ont des approches de test et des activités de test spécifiques, par exemple, dans la stratégie d'Automatisation des tests, les tests de performance, les tests de sécurité, les tests d'IA et les tests d'applications mobiles, ou lorsque des connaissances spécifiques au domaine sont requises (par exemple, les tests de logiciels automobiles ou les tests de jeux). Veuillez consulter le site www.istqb.org pour obtenir les dernières informations sur le programme de testeur certifié de l'ISTQB®.

0.4 Objectifs métier

Cette section énumère les objectifs métier attendus d'un candidat ayant obtenu la certification en Automatisation des tests - Ingénierie.

Un candidat ayant obtenu la certification en Automatisation des tests peut…

TAE-B01	Décrire l'objectif de l'Automatisation des tests.
TAE-B02	Comprendre l'Automatisation des tests à travers le cycle de vie du développement logiciel.
TAE-B03	Comprendre la configuration d'une infrastructure pour permettre l'Automatisation des tests.
TAE-B04	Apprendre le processus d'évaluation pour sélectionner les bons outils et les bonnes stratégies.
TAE-B05	Comprendre les préceptes de conception pour concevoir des solutions d'Automatisation des tests modulaires et évolutives.
TAE-B06	Choisir une approche, y compris un pilote, pour planifier le déploiement de l'Automatisation des tests dans le cycle de vie du développement logiciel.
TAE-B07	Concevoir et développer des solutions d'Automatisation des tests (nouvelles ou modifiées) qui répondent aux besoins techniques.
TAE-B08	Considérer le périmètre et l'approche de l'Automatisation des tests et de la maintenance des testware.

TAE-B09	Comprendre comment les tests automatisés s'intègrent dans les pipelines CI/CD.
TAE-B10	Comprendre comment collecter, analyser et faire des rapports sur les données d'Automatisation des tests afin d'informer les parties prenantes.
TAE-B11	Vérifier l'infrastructure d'Automatisation des tests.
TAE-B12	Définir les opportunités d'amélioration continue pour l'Automatisation des tests.

0.5 Objectifs d'apprentissage examinables et niveau cognitif de connaissance

Les objectifs d'apprentissage soutiennent les objectifs métier et sont utilisés pour créer les examens de Testeur certifié Automatisation des tests - Ingénierie.

En général, tous les contenus de ce syllabus sont examinables aux niveaux K2, K3 et K4, à l'exception de l'Introduction et des Annexes. En d'autres termes, le candidat peut être amené à reconnaître, mémoriser ou rappeler un mot-clé ou un concept mentionné dans l'un des huit chapitres. Les niveaux des objectifs d'apprentissage spécifiques sont indiqués au début de chaque chapitre et classés comme suit :

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

Des détails supplémentaires et des exemples d'objectifs d'apprentissage sont donnés dans l'annexe A.

Tous les termes listés comme mots-clés juste en dessous des titres de chapitres doivent être retenus, même s'ils ne sont pas explicitement mentionnés dans les objectifs d'apprentissage.

0.6 L'examen d'Automatisation des tests - ingénierie

L'examen du certificat d'Automatisation des tests - Ingénierie est basé sur ce syllabus. Les réponses aux questions de l'examen peuvent nécessiter l'utilisation d'exigences basées sur plus d'une section de ce syllabus. Toutes les sections du syllabus sont examinables, à l'exception de l'introduction et des annexes. Les normes et les livres sont inclus comme références, mais leur contenu n'est pas examinable, au-delà de ce qui est résumé dans le syllabus lui-même à partir de ces normes et de ces livres.

Pour plus de détails concernant l'examen de certification en Automatisation des tests - Ingénierie, se référer au document Structures et règles d'examen v1.1 compatibles avec les syllabus de niveaux Fondation et Avancé et les modules Spécialistes.

Le critère d'entrée pour passer l'examen d'Automatisation des tests - Ingénierie est que les candidats aient un intérêt pour les tests de logiciels et l'Automatisation des tests. Cependant, il est fortement recommandé aux candidats de

- Disposer d'une expérience minimale en matière de développement et de test de logiciels, par exemple six mois d'expérience en tant qu'ingénieur de test de logiciels ou en tant que développeur de logiciels.
- Suivre un cours accrédité conçu selon les recommandations de l'ISTQB® (accrédité par l'un des Membres reconnus par l'ISTQB®).

Note sur les exigences d'admission : le certificat de niveau Fondation de l'ISTQB® doit être obtenu avant de passer l'examen de certification en Automatisation des tests de l'ISTQB®.

0.7 Accréditation

Un Membre de l'ISTQB® peut accréditer les organismes de formation dont le matériel de cours suit ce syllabus. Les organismes de formation doivent obtenir les directives d'accréditation auprès du Membre ou de l'organisme qui effectue l'accréditation. Un cours accrédité est reconnu comme étant conforme à ce syllabus et peut comporter un examen de l'ISTQB®.

Les directives d'accréditation pour ce syllabus suivent les directives générales d'accréditation publiées par le groupe de travail sur la Gestion des Processus et la conformité.

0.8 Gestion des normes

Certaines normes sont référencées dans le syllabus de l'Automatisation des tests - Ingénierie (par exemple, IEEE et ISO). Le but de ces références est de fournir un framework (comme dans les références à ISO 25010 concernant les caractéristiques de qualité) ou de fournir une source d'information supplémentaire si le lecteur le souhaite. Veuillez noter que le syllabus utilise les documents de normes comme référence. Les documents de normes ne sont pas destinés à l'examen. Pour plus d'informations sur les normes, reportez-vous au chapitre 9 "Références".

0.9 Actualisation

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

0.10 Niveau de détail

Le niveau de détail de ce syllabus permet des cours et des examens cohérents au niveau international. Afin d'atteindre cet objectif, le syllabus se compose :

- Des objectifs pédagogiques généraux décrivant l'intention de l'ingénieur en Automatisation des tests de niveau Spécialiste.
- Une liste de termes que les étudiants doivent être en mesure de rappeler.
- Des objectifs d'apprentissage pour chaque domaine de connaissance, décrivant le résultat d'apprentissage cognitif à atteindre.
- Une description des concepts clés, y compris des références à des sources telles que la littérature ou les normes reconnues.

Le contenu du syllabus n'est pas une description de l'ensemble du domaine de connaissance des tests de logiciels ; il reflète le niveau de détail à couvrir dans les cours de formation en Automatisation des tests. Il se concentre sur les concepts et techniques de test qui peuvent s'appliquer à tous les projets logiciels, y compris ceux qui suivent les méthodes Agile. Ce syllabus ne contient pas d'objectifs d'apprentissage spécifiques liés aux tests en mode Agile, mais il aborde la façon dont ces concepts

s'appliquent dans les projets en mode Agile et dans d'autres types de tests.

0.11 Organisation de ce syllabus

Il y a huit chapitres dont le contenu peut faire l'objet d'un examen. L'en-tête de chaque chapitre précise la durée du chapitre ; le calendrier n'est pas fourni au-dessous du niveau du chapitre. Pour les formations accréditées, le syllabus exige un minimum de 21 heures d'enseignement, réparties sur les huit chapitres comme suit :

- Chapitre 1 : 45 minutes - Introduction et objectifs de l'Automatisation des tests.
  - Le testeur apprendra les avantages de l'Automatisation des tests et ses limites.
  - L'Automatisation des tests dans le cadre de différents modèles de cycle de vie du développement logiciel sera couverte.
  - Le testeur apprendra comment l'architecture d'un système sous test (SUT) a un impact sur l'adéquation des outils de test.
- Chapitre 2 : 180 minutes - Se préparer à l'Automatisation des tests
  - La conception pour la testabilité du SUT à travers l'observabilité, la contrôlabilité, et une architecture clairement définie, seront couvertes.
  - Le testeur apprendra l'Automatisation des tests dans différents environnements.
  - Les exigences nécessaires à l'audit d'une solution d'Automatisation des tests appropriée seront couvertes.
  - Le testeur apprendra les considérations techniques nécessaires pour développer des recommandations sur l'Automatisation des tests.
- Chapitre 3 : 210 minutes - Architecture d'Automatisation des tests
  - L'architecture d'Automatisation des tests et ses composants menant à une solution d'Automatisation des tests seront couverts.
  - Le testeur apprendra à connaître les couches et leur application dans un framework d'Automatisation des tests.
  - De multiples approches de l'utilisation des outils d'Automatisation des tests seront couvertes.
  - Le testeur apprendra comment les principes de conception et les canevas de conception peuvent être appliqués à l'Automatisation des tests.
- Chapitre 4 : 150 minutes - Implémentation de l'Automatisation des tests.
  - La planification et le déploiement d'un projet pilote d'Automatisation des tests seront couverts.
  - Le testeur apprendra les risques de déploiement et les stratégies d'atténuation.
  - Les facteurs qui améliorent la maintenabilité du code d'Automatisation des tests seront couverts.
- Chapitre 5 : 90 minutes - Stratégies d'implémentation et de déploiement de l'Automatisation des tests.

  - Le testeur en apprendra davantage sur les pipelines CI/CD et l'exécution automatisée des tests à travers les niveaux de test.
  - La gestion de la configuration pour les composants de l'Automatisation des tests sera couverte.
  - Le testeur apprendra les dépendances appliquées aux tests d'API et de contrats.
- Chapitre 6 : 150 minutes - Automatisation des tests : rapports et métriques
  - Le testeur apprendra où les données peuvent être collectées à partir d'un SUT et l'Automatisation des tests pour l'analyse et le reporting des tests.
  - L'analyse des données des rapports sur le SUT et l'Automatisation des tests pour découvrir les causes des défaillances seront couvertes.
  - L'utilisation des rapports de test et des tableaux de bord pour informer les parties prenantes sera couverte.
- Chapitre 7 : 135 minutes - Vérifier la solution d'Automatisation des tests.
  - Le testeur apprendra à examiner et à vérifier le bon fonctionnement des composants et de l'environnement d'Automatisation des tests.
  - Les mesures pour s’assurer que les scripts de test et les suites de tests s'exécutent correctement seront couvertes.
  - Le testeur comprendra quand effectuer une analyse des causes racines
  - Les techniques d'analyse de la qualité du code d'Automatisation des tests seront couvertes.
- Chapitre 8 : 210 minutes - Amélioration continue
  - D'autres domaines d'analyse de données pour l'amélioration des cas de test seront couverts.
  - Le testeur apprendra des façons d'apporter des améliorations et des mises à niveau à une solution d'Automatisation des tests et à ses composants.
  - L'identification et les moyens de consolider et de rationaliser l'Automatisation des tests seront couverts.
  - Le testeur apprendra comment les outils d'Automatisation des tests peuvent contribuer au soutien et à la mise en place des tests.

1 Introduction et objectifs de l'Automatisation des tests – 45 minutes (K-2)

Mots clés

système sous test, Automatisation des tests, ingénieur en Automatisation des tests

Objectifs d'apprentissage pour le chapitre 1 :

1.1 Objectif de l'Automatisation des tests

TAE-1.1.1 (K2) Expliquer les avantages et les inconvénients de l'Automatisation des tests.

1.2 L'Automatisation des tests dans le cycle de vie du développement logiciel

TAE-1.2.1 (K2) Expliquer comment l'Automatisation des tests est appliquée dans les différents modèles de cycle de vie du développement logiciel.

TAE-1.2.2 (K2) Sélectionner les outils d'Automatisation des tests appropriés pour un système sous test donné.

1.1 Objectif de l'Automatisation des tests

1.1.1 Expliquer les avantages et les inconvénients de l'Automatisation des tests

L'Automatisation des tests, qui comprend l'exécution automatisée des tests et l'établissement de rapports sur les tests, correspond à une ou plusieurs des activités suivantes :

- Utilisation d'outils logiciels conçus à cet effet pour contrôler et mettre en place des suites de tests pour l'exécution des tests.
- Exécution de tests de manière automatisée.
- Comparaison des résultats réels aux résultats attendus.

L'Automatisation des tests offre des caractéristiques et des capacités significatives qui peuvent interagir avec un système sous test (SUT). L'Automatisation des tests peut couvrir un large domaine des logiciels. Les solutions couvrent de nombreux types de logiciels (par exemple, SUT avec une interface utilisateur (UI), SUT sans UI, applications mobiles, protocoles réseau et connexions).

L'Automatisation des tests présente de nombreux avantages. Elle :

- Permet d'exécuter plus de tests par version (build) par rapport aux tests manuels.
- Offre la possibilité de créer et d'exécuter des tests qui ne peuvent pas être exécutés manuellement (par exemple, réactivité en temps réel, tests à distance et tests en parallèle).
- Permet de réaliser des tests plus complexes que les tests manuels.
- S'exécute plus rapidement que les tests manuels.
- Est moins sujette à l'erreur humaine.
- Est plus efficace et efficiente dans l'utilisation des ressources de test.
- Fournit un retour d'information plus rapide concernant la qualité du SUT.
- Contribue à améliorer la fiabilité du système (par exemple, la disponibilité et la récupération).
- Améliore la cohérence de l'exécution des tests sur l'ensemble des cycles de test.

Cependant, l'Automatisation des tests présente des inconvénients potentiels, notamment :

- Des coûts supplémentaires seront impliqués pour le projet car il peut être nécessaire d'engager un ingénieur en Automatisation des tests (TAE), d'acheter du nouveau matériel et de mettre en place une formation.
- La nécessité d'un investissement initial pour mettre en place une solution d'Automatisation des tests.
- Le temps nécessaire pour développer et maintenir une solution d'Automatisation des tests.
- Le besoin d'objectifs clairs en matière d'Automatisation des tests pour garantir le succès.
- La rigidité des tests, et moins d'adaptabilité aux changements dans le SUT.
- L'introduction de défauts supplémentaires dus à l'Automatisation des tests.

L'Automatisation des tests présente des limites qu'il faut garder à l'esprit :

- Tous les tests manuels ne peuvent pas être automatisés.
- Ne vérifie que ce que les tests automatisés sont programmés pour faire.
- L'Automatisation des tests ne peut vérifier que les résultats des tests interprétables par une machine, ce qui signifie que certaines caractéristiques de qualité peuvent ne pas être testables avec l'automatisation.
- L'Automatisation des tests ne peut vérifier que les résultats du test qui peuvent être vérifiés par un oracle de test automatisé.

1.2 L'automatisation dans le cycle de vie de développement logiciel

1.2.1 Expliquer comment l'Automatisation des tests est appliquée dans les différents modèles de cycles de vie de développement logiciel ?

Cascade

Le modèle en cascade est un modèle SDLC à la fois linéaire et séquentiel. Ce modèle comporte des phases distinctes (exigences, conception, implémentation, vérification et maintenance) et chaque phase se termine généralement par une documentation qui doit être approuvée. L'implémentation de l'Automatisation des tests se fait généralement en parallèle ou après la phase d'implémentation. Les exécutions de test ont généralement lieu pendant la phase de vérification, car les composants logiciels ne sont pas prêts à être testés avant ce moment-là.

Modèle en V

Le modèle en V est un modèle de cycle de vie de développement logiciel dans lequel un processus est exécuté de manière séquentielle. Comme un projet est défini depuis les exigences de haut niveau jusqu'aux exigences de bas niveau, les activités de test et d'intégration correspondantes sont définies pour valider ces exigences. C'est de là que découlent les niveaux de test traditionnels : composant, intégration des composants, système, intégration des systèmes et acceptation, comme décrit dans la section 2.2 du syllabus de niveau Fondation. Il est possible et recommandé de fournir un framework d'Automatisation des tests (TAF) pour chaque niveau de test.

Développement de logiciel en mode Agile

Dans le développement logiciel en mode Agile, les possibilités d'Automatisation des tests sont innombrables. Contrairement à la cascade ou au modèle V, dans la méthode de développement logiciel en mode Agile, les TAE et les représentants du métier peuvent décider de la feuille de route, du calendrier et de la planification des tests. Dans ces méthodes, il existe de meilleures pratiques telles que les revues de code, la programmation en binôme et l'exécution fréquente de tests automatisés. L'élimination des silos (c'est-à-dire en s'assurant que les développeurs, les testeurs et les autres parties prenantes travaillent ensemble) permet aux équipes de couvrir tous les niveaux de test avec la quantité et la profondeur appropriées d'automatisation, atteignant ainsi un objectif appelé automatisation in-sprint. De plus amples renseignements figurent à la section 3.2. du syllabus CT-TAS (Stratégie d'Automatisation des tests) de l'ISTQB®.

1.2.2 Choisir des outils d'Automatisation des tests adaptés à un système sous test donné

Pour identifier les outils de test les plus adaptés à un projet donné, le SUT doit d'abord être analysé. Les TAE doivent identifier les exigences du projet qui peuvent être utilisées comme base pour la sélection des outils.

Étant donné que différentes caractéristiques des outils d'Automatisation des tests sont utilisées pour les logiciels d'interface utilisateur et, par exemple, les services web, il est important de comprendre ce que le projet veut atteindre au fil du temps. Il n'y a pas de limite au nombre d'outils d'Automatisation des tests et de caractéristiques qui peuvent être utilisés ou sélectionnés, mais les coûts doivent toujours être pris en compte. L'utilisation d'un outil commercial sur étagère ou l'implémentation d'une solution personnalisée basée sur une technologie open-source peut être un processus complexe.

Le sujet suivant à évaluer est la composition et l'expérience de l'équipe en matière d'Automatisation des tests. Dans le cas où les testeurs ont peu ou pas d'expérience en programmation, l'utilisation d'une solution « low-code » ou « no-code » peut être un choix viable.

Pour les testeurs techniques ayant des connaissances en programmation, il peut être utile de choisir des outils dont le langage correspond à celui du SUT. Cela présente des avantages, notamment la possibilité de travailler avec les développeurs sur le débogage des défauts d'Automatisation des tests de manière plus efficiente et la formation conjointe des membres de différentes équipes.

2 Se préparer à l'Automatisation des tests – 180 minutes (K4)

Mots clés

Tests d'API, tests de l'interface graphique, testabilité

Objectifs d'apprentissage pour le chapitre 2:

2.1 Comprendre la configuration d'une infrastructure permettant l'Automatisation des tests

TAE-2.1.1 (K2) Décrire les besoins de configuration d'une infrastructure permettant l'implémentation de l'Automatisation des tests.

TAE-2.1.2 (K2) Expliquer comment l'Automatisation des tests est exploitée dans différents environnements.

2.2 Processus d'évaluation pour sélectionner les bons outils et les bonnes stratégies

TAE-2.2.1 (K4) Analyser un système sous test pour déterminer la solution d'Automatisation des tests appropriée.

TAE-2.2.2 (K4) Illustrer les constatations techniques d'une évaluation d'outil.

2.1 Comprendre la configuration d'une infrastructure permettant l'Automatisation des tests

2.1.1 Décrire les besoins de configuration d'une infrastructure permettant l'implémentation de l'Automatisation des tests

La testabilité du SUT (c'est-à-dire la disponibilité d'interfaces logicielles qui soutiennent les tests, par exemple pour permettre le contrôle et l'observabilité du SUT) devrait être conçue et implémentée parallèlement à la conception et à l'implémentation des autres caractéristiques du SUT. Ce travail est généralement effectué par un architecte logiciel car la testabilité est une exigence non fonctionnelle du système. Il est souvent accompagné d'un TAE pour identifier les domaines spécifiques où des améliorations peuvent être apportées.

Pour une meilleure testabilité du SUT, il existe différentes solutions qui peuvent être utilisées et qui ont des besoins de configuration différents, par exemple :

- Identifiants d'accessibilité
  - Les différents frameworks de développement peuvent générer ces identifiants automatiquement ou les développeurs peuvent les définir manuellement.
- Variables d'environnement du système
  - Certains paramètres de l'application peuvent être modifiés pour permettre de tester plus facilement l'application par le biais de l'administration.
- Variables de déploiement
  - Similaires aux variables du système, mais qui peuvent être définies avant de commencer le déploiement.

La conception de la testabilité d'un SUT comprend les aspects suivants :

- Observabilité : Le SUT doit fournir des interfaces qui donnent un aperçu du SUT. Les cas de test peuvent alors utiliser ces interfaces pour déterminer si les résultats réels correspondent aux résultats attendus.
- Contrôlabilité : Le SUT doit fournir des interfaces qui peuvent être utilisées pour effectuer des actions sur celui-ci. Il peut s'agir d'éléments d'interface utilisateur, d'appels de fonction, d'éléments de communication (par exemple, protocole de contrôle de transmission/protocole Internet (TCP/IP) et protocoles de bus série universel (USB)) ou de signaux électroniques pour des commutateurs physiques ou logiques sur les différentes variables d'environnement.
- Transparence de l'architecture : La documentation d'une architecture doit fournir des composants et des interfaces clairs et compréhensibles qui donnent une observabilité et un contrôle à tous les niveaux de test et favorisent la qualité.

2.1.2 Expliquer comment l'Automatisation des tests est exploitée au sein de différents environnements.

Différents types de tests automatisés peuvent être exécutés dans différents environnements. Ces environnements peuvent différer selon les projets et les méthodologies, et la plupart des projets disposent d'un ou plusieurs environnements utilisés pour tester. D'un point de vue technique, ces environnements peuvent être créés à partir de conteneurs, de logiciels de virtualisation et d'autres approches.

Voici une série d'environnements possibles à prendre en considération :

Environnement de développement local

L'environnement de développement local est l'endroit où le logiciel est initialement créé et où les composants sont testés par automatisation pour vérifier leur aptitude fonctionnelle. Plusieurs types de tests peuvent être effectués dans l'environnement de développement local, notamment des tests de composants, des tests d'interface graphique et des tests d'interface de programmation d'applications (API). Il est également important de noter qu'en utilisant un environnement de développement intégré (IDE) sur l'ordinateur donné, des tests boîte blanche peuvent être effectués afin d'identifier le plus tôt possible les problèmes de codage et de qualité médiocre.

Environnement de Build

Son objectif principal est de construire le logiciel (NDT: build) et d'exécuter des tests qui vérifient l'exactitude de la construction résultante dans un écosystème DevOps. Cet environnement peut être soit un environnement de développement local, soit un agent d'intégration continue/de livraison continue (CI/CD) où les tests de bas niveau (c'est-à-dire les tests de composants et les tests d'intégration continue des composants) et l'analyse statique peuvent être effectués sans déploiement réel dans d'autres environnements.

Environnement d'intégration

Après les tests de bas niveau et l'analyse statique, l'étape suivante est un environnement d'intégration des systèmes. Il s'agit d'un candidat à la release du SUT qui est entièrement intégré à d'autres systèmes pouvant être testés. Dans cet environnement, une suite de tests entièrement automatisée, soit des tests d'interface utilisateur, soit des tests d'API, peut être exécutée. Dans cet environnement, il n'y a pas de tests boîte blanche, mais uniquement des tests boîte noire (c'est-à-dire des tests d'intégration des systèmes et/ou des tests d'acceptation). Il est important de noter qu'il s'agit du premier environnement où une surveillance (NDT : monitoring) devrait être présente pour voir ce qui se passe en arrière-plan pendant l'utilisation du SUT afin de permettre une investigation efficiente des défauts/défaillances.

Environnement de préproduction

Un environnement de préproduction est utilisé principalement pour auditer les caractéristiques de qualité non fonctionnelles (par exemple, l'efficience de la performance). Bien que les tests non fonctionnels puissent être réalisés dans n'importe quel environnement, l'accent est mis sur la préproduction parce qu'elle ressemble le plus possible à la production. Souvent, les tests d'acceptation des utilisateurs peuvent être effectués par les parties prenantes Métier pour vérifier le produit final et il est possible d'exécuter la suite automatisée de tests existante ici aussi, si nécessaire. Cet environnement est également surveillé.

Environnement de production/d'exploitation

Un environnement de production peut être utilisé pour évaluer les caractéristiques de qualité fonctionnelles et non fonctionnelles en temps réel pendant que les utilisateurs interagissent avec un système déployé grâce au suivi et à certaines meilleures pratiques qui permettent les tests en production (par exemple, la release canari, le déploiement bleu/vert et les tests A/B).

2.2 Processus d'évaluation pour sélectionner les bons outils et stratégies

2.2.1 Analyser un système sous test pour déterminer la solution d'Automatisation des tests appropriée

Chaque SUT peut être différent d'un autre, mais il y a plusieurs facteurs et caractéristiques qui peuvent être analysés pour avoir une solution d'Automatisation des tests (TAS) réussie. Au cours de l'étude d'un SUT, les TAE doivent rassembler les exigences en tenant compte de son périmètre et de ses capacités données. Différents types d'applications (par exemple, service Web, mobile et Web) nécessitent différents types d'Automatisation des tests d'un point de vue technique. L'enquête peut être effectuée - c'est recommandé - en collaboration avec d'autres parties prenantes (par exemple, les testeurs manuels, les parties prenantes métier et les analystes métier) pour identifier autant de risques et leurs atténuations que possible afin d'avoir une solution d'Automatisation des tests bénéfique pour l'avenir.

Les exigences d'une approche d'Automatisation des tests et d'une architecture d'Automatisation des tests doivent prendre en compte les éléments suivants :

- Quelles activités du processus de test doivent être automatisées, (par exemple, la gestion des tests, la conception des tests, la génération des tests et l'exécution des tests).
- Quels niveaux de test doivent être soutenus ?
- Quels types de tests doivent être soutenus ?
- Quels rôles et compétences en matière de tests doivent être soutenus ?
- Quels produits, lignes de produits et familles de logiciels doivent être soutenus (par exemple, pour définir l'étendue et la durée de vie du SAE implémenté ?
- Quels types de SUT doivent être compatibles avec la TAS ?
- Quelle est la disponibilité des données de test et quelle est leur qualité ?
- Quelles sont les méthodes possibles et les moyens d'émuler des cas inaccessibles (par exemple, les applications tierces concernées) ?

2.2.2 Illustrer les constatations techniques d'une évaluation d'outil

Après l'analyse du SUT et la collecte des exigences auprès de toutes les parties prenantes, il est probable que des outils d'Automatisation des tests répondant à ces exigences puissent être envisagés. Il se peut qu'il n'y ait pas un seul outil qui réponde à toutes les exigences identifiées, et les parties prenantes devraient reconnaître cette possibilité.

Il est utile de rassembler les constatations sur les outils possibles et de réfléchir aux diverses exigences directes et indirectes dans un tableau comparatif. L'objectif du tableau de comparaison est de permettre aux parties prenantes de voir les différences entre les outils sur la base d'exigences spécifiques. Le tableau de comparaison présente les outils dans les colonnes et les exigences dans les lignes. Les cellules contiennent des informations sur les propriétés de chaque outil par rapport à chaque exigence ainsi que sur les priorités.

En général, les outils d'Automatisation des tests doivent être évalués pour déterminer s'ils répondent à l'exigence identifiée dans la section précédente (2.2.1). Les exigences à prendre en compte lors de l'évaluation et de la comparaison des outils comprennent :

- Le langage/la technologie de l'outil et les outils IDE.
- La capacité à configurer un outil, qu'il soutienne différents environnements de test, qu'il exécute des configurations et qu'il utilise des valeurs de configuration dynamiques ou statiques.
- La capacité à gérer les données de test au sein de l'outil. La gestion des données de test pourrait être intégrée à un référentiel central pour le contrôle des versions.
- La nécessité éventuelle de devoir sélectionner différents outils d'Automatisation des tests pour différents types de tests.
- La capacité à fournir des fonctionnalités de reporting. Ceci est important pour s'aligner sur les exigences du projet en matière de reporting des tests.
- La capacité à s'intégrer à d'autres outils utilisés sur un projet ou dans l'organisation, tels que CI/CD, le suivi des tâches, la gestion des tests, le reporting ou d'autres outils.
- La capacité d'étendre l'architecture de test globale et d'évaluer l'évolutivité, la maintenabilité, la facilité de modification, la compatibilité et la fiabilité des outils.

Ce tableau de comparaison est une bonne source pour déterminer une proposition d'outil ou d'ensemble d'outils à utiliser pour l'Automatisation des tests du SUT.

Le processus peut varier quant à la manière dont la décision est prise sur le(s) outil(s) à utiliser, mais la proposition doit être présentée aux parties prenantes appropriées pour approbation.

3 Architecture d'Automatisation des tests – 210 minutes (K3)

Mots clés

développement piloté par les comportements, capture/rejeux, tests génériques d'Automatisation des tests, architecture d'Automatisation des tests, tests pilotés par les mots-clés, scripting linéaire, tests basés sur des modèles, scripting structuré, couche d'adaptation des tests, architecture d'Automatisation des tests, harnais de tests, test piloté par les données, étape de test, solution d'Automatisation des tests, développement piloté par les tests, script de test

Objectifs d'apprentissage pour le chapitre 3 :

3.1 Concepts de conception utilisés dans l'Automatisation des tests

TAE-3.1.1 (K2) Expliquer les principales fonctionnalités d'une architecture d'Automatisation des tests.

TAE-3.1.2 (K2) Expliquer comment concevoir une solution d'Automatisation des tests.

TAE-3.1.3 (K3) Appliquer la stratification des frameworks d'Automatisation des tests.

TAE-3.1.4 (K3) Appliquer différentes approches pour l'automatisation des cas de test.

TAE-3.1.5 (K3) Appliquer les principes et les modèles de conception dans l'Automatisation des tests.

3.1 Concepts de design exploités dans l'automatisation des tests

3.1.1 Expliquer les principales capacités d'une architecture d'Automatisation des tests

Architecture générique d'Automatisation des tests (gTAA = TAA générique)

La gTAA est un concept de conception de haut niveau qui fournit une vue abstraite de la communication entre l'Automatisation des tests et les systèmes auxquels l'Automatisation des tests est connectée, c'est-à-dire le SUT, la gestion de projet, la gestion des tests et la gestion de la configuration (voir figure 1). Il fournit également les capacités qu'il est nécessaire de couvrir lors de la conception d'une architecture d'Automatisation des tests (TAA).

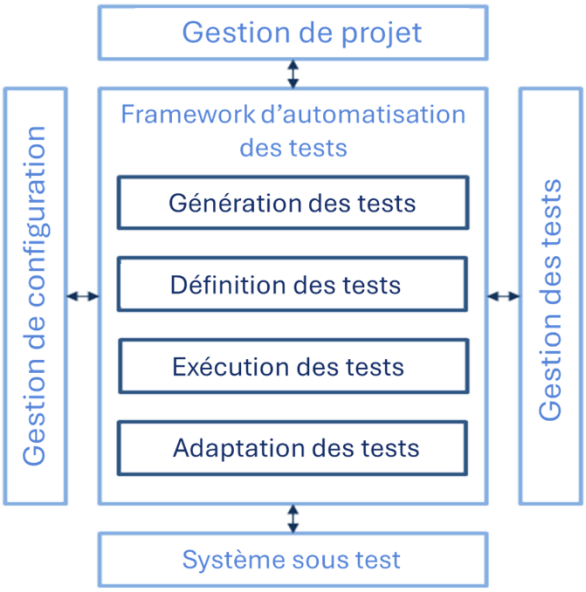

Figure 1: diagramme gTAA

Les interfaces de gTAA décrivent les éléments suivants :

- L'interface SUT décrit la connectivité entre le SUT et le TAF (voir la section 3.1.3 concernant le framework d'Automatisation des tests).
- L'interface de gestion du projet décrit l'avancement du développement de l'Automatisation des tests.
- L'interface de management des tests décrit la cartographie des définitions de cas de test et des cas de test automatisés.
- L'interface de gestion de la configuration décrit les pipelines CI/CD, les environnements et le testware.

Capacités offertes par les outils et bibliothèques d'Automatisation des tests

Les principales capacités d'Automatisation des tests doivent être identifiées et sélectionnées parmi les outils disponibles en fonction des exigences d'un projet donné.

Génération de tests : Soutient la conception automatisée des cas de tests basés sur des modèles de tests. Les tests basés sur des modèles peuvent être utilisés dans le processus de génération (voir le syllabus CT-MBT de l'ISTQB). La génération de tests est une capacité optionnelle**.**

Définition des tests : Soutient la définition et l'implémentation des cas de test et/ou des suites de tests, qui peuvent éventuellement être dérivés d'un modèle de test. Elle sépare la définition du test du SUT et/ou des outils de test. Elle contient les moyens de définir des tests de haut niveau et de bas niveau, qui sont traités dans les données de test, les cas de test et les composants de la bibliothèque de test ou des combinaisons de ceux-ci.

Exécution des tests : Soutient l'exécution du test et le logging des tests. Il fournit un outil d'exécution des tests pour exécuter automatiquement les tests sélectionnés, ainsi qu'un composant de logging des tests et de reporting des tests.

Adaptation des tests : Fournit la fonctionnalité nécessaire pour adapter les tests automatisés aux différents composants ou interfaces du SUT. Il fournit différents adaptateurs pour se connecter au SUT via des API, des protocoles et des services.

3.1.2 Expliquer comment concevoir une solution d'Automatisation des tests

Une solution d'Automatisation des tests (TAS) est définie par une compréhension des exigences fonctionnelles, non fonctionnelles et techniques du SUT, des outils existants ou requis qui sont nécessaires pour implémenter une solution. Une solution d'Automatisation des tests est implémentée avec des outils commerciaux ou open-source et peut nécessiter des adaptateurs supplémentaires spécifiques au SUT.

Le TAA définit la conception technique de l'ensemble du TAS. Elle doit aborder :

- La spécification des outils d'Automatisation des tests et des bibliothèques spécifiques aux outils.
- Le développement d'extensions et/ou de composants.
- L'identification des exigences en matière de connectivité et d'interface (par exemple, pare-feu, base de données, localisateurs de ressources uniformes (URL)/connexions, mocks/stubs, files d'attente de messages et protocoles).
- La connexion aux outils de gestion des tests et des défauts.
- L'utilisation d'un système de contrôle de version et de référentiels.

3.1.3 Appliquer une superposition dans les frameworks d'Automatisation des tests

Le framework d'Automatisation des tests

Le TAF est la base d'un TAS. Il comprend souvent un harnais de test, également connu sous le nom de lanceur de test, ainsi que des bibliothèques de test, des scripts de test et des suites de tests.

Couches du TAF

Les couches du TAF définissent une frontière distincte de classes qui ont des objectifs similaires tels que les cas de test, le reporting des tests, le logging des tests, le cryptage et les harnais de test. En introduisant une couche pour chaque objectif unique, la conception peut devenir compliquée. Il est donc recommandé de limiter le nombre de couches du TAF.

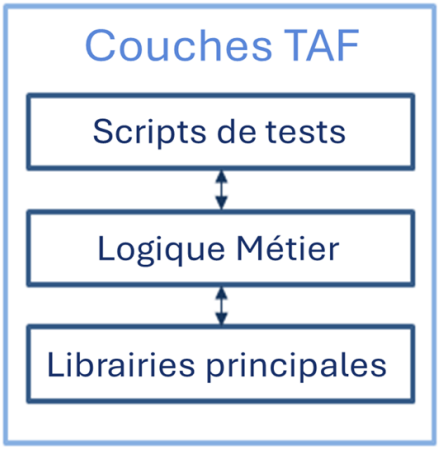

Figure 2: Couches du framework d'automatisation des tests (TAF)

Les scripts de test

L'objectif de cette couche est de fournir un référentiel de cas de test du SUT et des annotations de la suite de tests. Elle appelle les services de la couche logique métier, ce qui peut impliquer des étapes de test, des flux d'utilisateurs ou des appels d'API. Toutefois, aucun appel direct aux bibliothèques principales ne doit être effectué à partir des scripts de test.

Logique métier

Toutes les bibliothèques dépendantes du SUT sont stockées dans cette couche. Ces bibliothèques hériteront des fichiers de classe des bibliothèques principales ou utiliseront les façades fournies par celles-ci (voir la section 3.1.5 concernant l'héritage et les façades). La couche de logique métier est utilisée pour configurer le TAF afin qu'il s'exécute au regard du SUT et les configurations supplémentaires.

Bibliothèques principales

Toutes les bibliothèques indépendantes de tout SUT sont stockées dans cette couche. Ces bibliothèques principales peuvent être réutilisées dans n'importe quel type de projet partageant la même pile de développement.

Mise à l'échelle de l'Automatisation des tests

L'exemple suivant (figure 3) montre comment les bibliothèques principales fournissent une base réutilisable pour plusieurs TAF. Dans le projet #1, il y a deux TAFs construits au-dessus des bibliothèques principales, et un projet séparé utilise les bibliothèques principales déjà existantes pour construire son TAF afin de tester l'application #3. Un TAE construit les TAFs pour le projet #1, tandis qu'un second TAE construit le TAF pour le projet #2.

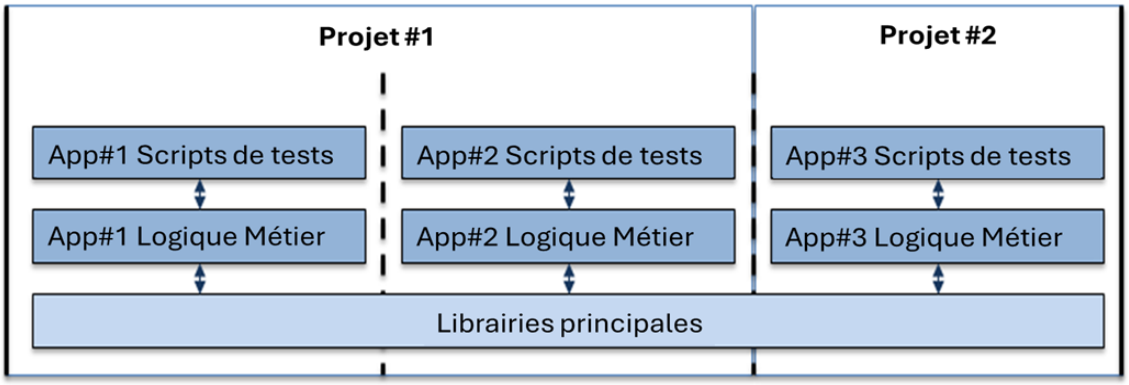

Figure 3: Application des bibliothèques principales à des TAFs multiples

3.1.4 Appliquer différentes approches pour automatiser les cas de tests

Il existe plusieurs approches de développement que les équipes peuvent choisir pour produire des cas de tests automatisés. Il peut s'agir de langages de script interactifs ou de langages de programmation compilés. Les différentes approches offrent des avantages différents en matière d'automatisation et peuvent être exploitées dans des circonstances différentes. Bien que le développement piloté par les tests (TDD) et le développement piloté par le comportement (BDD) soient des méthodologies de développement, si elles sont suivies correctement, elles ont pour résultat le développement automatisé de cas de test.

Capture/rejeux

La capture/rejeux est une approche qui capture les interactions avec le SUT pendant qu'une séquence d'actions est exécutée manuellement. Ces outils produisent des scripts de test pendant la capture, et selon l'outil utilisé, le code d'Automatisation des tests peut être modifiable. Les outils qui n'exposent pas de code sont parfois appelés Automatisation des tests no-code, tandis que les outils exposant du code

sont appelés Automatisation des tests low-code.

Avantages

- Initialement facile à mettre en place et à utiliser

Inconvénients

- Difficile à maintenir, à mettre à l'échelle et à faire évoluer.
- Le SUT doit être disponible lors de la capture d'un cas de test.
- Uniquement réalisable pour un périmètre restreint et un SUT qui change rarement.
- L'exécution du SUT capturé dépend fortement de la version du SUT à partir de laquelle la capture a été effectuée.
- Enregistrer chaque cas de test individuel au lieu de réutiliser les blocs de construction existants prend du temps.

Scripting linéaire

Le script linéaire est une activité de programmation qui ne nécessite pas de bibliothèques de test personnalisées réalisées par un TAE et qui est utilisée pour écrire et exécuter les scripts de test. Un TAE peut s'appuyer sur des scripts de test enregistrés par un outil de capture/rejeux, qu'il peut ensuite modifier.

Avantages

- Facile à mettre en place et à commencer à écrire des scripts de test.
- Par rapport à la capture/rejeux, les scripts de test peuvent être modifiés plus facilement.

Inconvénients

- Difficile d'assurer la maintenance, la mise à l'échelle et l'évolution.
- L'utilisateur final doit être disponible lors de la capture d'un cas de test.
- Uniquement réalisable pour un petit périmètre et un SUT qui change rarement.
- Par rapport à la capture/rejeux, certaines connaissances en programmation sont nécessaires.

Scripting structuré

Des bibliothèques de test sont introduites avec des éléments réutilisables, des étapes de test et/ou des parcours utilisateurs. Des connaissances en programmation sont nécessaires pour la création et la maintenance des scripts de test dans cette approche.

Avantages

- Facilité de maintenabilité, de mise à l'échelle, de portage, d'adaptation et d'évolution.
- La logique métier peut être séparée des scripts de test.

Inconvénients

- Des connaissances en programmation sont nécessaires.
- L'investissement initial dans le développement du TAF et la définition du testware prend du temps.

Développement piloté par les tests (TDD)

Les cas de test sont définis dans le cadre du processus de développement avant qu'une nouvelle caractéristique du SUT ne soit implémentée. L'approche de développement piloté par les tests consiste à tester, coder et refactoriser, aussi connu sous le nom de "red, green, and refactor" (rouge, vert et refactoriser). Un développeur identifie et crée un cas de test qui échouera (rouge). Il développe ensuite une fonctionnalité qui satisfera le cas de test (vert). Le code est ensuite refactorisé afin de l'optimiser et de respecter les principes du code propre. Le processus se poursuit avec le test suivant et l'incrément de fonctionnalité suivant.

Avantages

- Simplifie le développement des cas de test au niveau des composants.
- Améliore la qualité et la structure du code.
- Améliore de la testabilité.
- Facilite l'obtention de la couverture de code souhaitée.
- Réduit la propagation des défauts à des niveaux de test plus élevés.
- Améliore la communication entre les développeurs, les représentants des métiers et les testeurs.
- Les User Stories qui ne sont pas vérifiées à l'aide des tests de l'interface graphique et des tests de l'API peuvent rapidement atteindre les critères de sortie en suivant la méthode TDD.

Inconvénients

- Au départ, il faut plus de temps pour s'habituer au TDD.
- Ne pas suivre correctement le TDD peut entraîner une fausse confiance dans la qualité du code.

Tests pilotés par les données

Les tests pilotés par les données (DDT) s'appuient sur l'approche du scripting structuré. Les scripts de test sont fournis avec des données de test (par exemple, des fichiers .csv, .xlsx et des vidages de base de données). Cela permet d'exécuter plusieurs fois les mêmes scripts de test avec des données de test différentes.

Avantages

- Permet d'étendre rapidement et facilement les cas de test grâce à des flux de données.
- Le coût de l'ajout de nouveaux tests automatisés peut être considérablement réduit.
- Les analystes de test peuvent spécifier des tests automatisés en alimentant un ou plusieurs fichiers de données de test qui décrivent les tests. Les analystes de test disposent ainsi d'une plus grande liberté pour spécifier des tests automatisés en dépendant moins des analystes techniques de test.

Inconvénients

- Une bonne gestion des données de test peut s'avérer nécessaire.

Tests pilotés par les mots-clés

Les tests pilotés par les mots-clés (KDT) sont une liste ou un tableau d'étapes de test dérivées de mots-clés et des données de test sur lesquelles les mots-clés opèrent. Les mots-clés sont définis du point de

vue de l'utilisateur. Cette technique est souvent basée sur le DDT.

Avantages

- Les analystes de test et les analystes métier peuvent être impliqués dans la création de cas de test automatisés en suivant l'approche du KDT.
- Les KDT peuvent également être utilisés pour les tests manuels indépendamment de l'Automatisation des tests (voir la norme ISO/IEC/IEEE 29119-5 concernant les tests pilotés par les mots-clés).

Inconvénients

- L'implémentation et la maintenabilité des mots-clés est une tâche complexe que les TAE doivent couvrir, ce qui peut devenir un défi lorsque le périmètre s'élargit.
- Cela engendre un effort considérable pour les systèmes de petite taille.

Développement piloté par le comportement (BDD)

BDD exploite un format de langage naturel (c.-à-d., étant donné, quand, alors (NDT: given, when, then)) pour formuler des critères d'acceptation qui peuvent être utilisés comme cas de tests automatisés et stockés dans des fichiers de caractéristiques. Un outil BDD peut alors comprendre le langage et exécuter les cas de test.

Avantages

- Améliore la communication entre les développeurs, les représentants du métier et les testeurs.
- Les scénarios BDD automatisés font office de cas de tests et assurent la couverture des spécifications.
- BDD peut être mis à profit pour produire plusieurs types de tests à différents niveaux de la pyramide des tests.

Inconvénients

- Des cas de test supplémentaires, généralement des conditions de test négatives et des cas limites doivent encore être définis par l'équipe, typiquement par un analyste de test ou un TAE.
- De nombreuses équipes confondent le BDD avec un simple moyen d'écrire des cas de test dans un langage naturel, et n'impliquent pas les représentants des métiers et les développeurs dans l'ensemble de l'approche.
- L'implémentation et la maintenance des étapes de test en langage naturel est une tâche complexe à couvrir pour les TAE.
- Des étapes de test trop complexes transformeront le débogage en une activité difficile et coûteuse.

3.1.5 Appliquer les principes et les canevas de conception à l'Automatisation des tests

L'Automatisation des tests est une activité de développement logiciel. Par conséquent, les principes et les canevas de conception sont tout aussi importants pour un TAE que pour un développeur de logiciels.

Principes de programmation orientée objet

Il existe quatre grands principes de programmation orientée objet : l'encapsulation, l'abstraction, l'héritage et le polymorphisme.

Principes SOLID

Il s'agit d'un acronyme de la respon**S**abilité unique, de l'**O**uverture-fermeture, de la substitution de **L**iskov, de la substitution d'**I**nterface et de l'inversion de **D**épendance (NDT : de l'anglais Single responsibility, Open-closed, Liskov substitution, Interface substitution and Dependency inversion). Ces principes améliorent la lisibilité du code, la maintenabilité et l'évolutivité.

Canevas de conception

Parmi les nombreux canevas de conception, trois sont plus importants pour les TAE.

Le modèle de façade masque les détails de l'implémentation pour n'exposer que ce que les testeurs doivent créer dans les cas de test, et le canevas de singleton est souvent utilisé pour s'assurer qu'il n'y a qu'un seul pilote qui communique avec le SUT.

Dans le modèle objet page, un fichier de classe est créé et appelé modèle de page. Chaque fois que la structure du SUT change, le TAE devra faire des mises à jour à un seul endroit, le localisateur à l'intérieur d'un modèle de page, au lieu de mettre à jour les localisateurs dans chaque cas de test.

Le canevas de modèle de flux est une extension du modèle d'objet de page. Il introduit une façade supplémentaire au-dessus des modèles d'objets de page, qui stocke toutes les actions de l'utilisateur qui interagissent avec les objets de page. En introduisant une conception à double façade, le canevas de modèle de flux améliore l'abstraction et la maintenabilité, car les étapes de test peuvent être réutilisées dans plusieurs scripts de test.

4 Implémentation de l'Automatisation des tests – 150 minutes (K4)

Mots clés

risque, contexte de test

Objectifs d'apprentissage pour le chapitre 4:

4.1 Développement de l'Automatisation des tests

TAE-4.1.1 (K3) Appliquer des lignes directrices qui soutiennent des activités efficaces d'Automatisation des tests en matière de pilotage et de déploiement.

4.2 Risques associés au développement de l'Automatisation des tests

TAE-4.2.1 (K4) Analyser les risques liés au déploiement et planifier des stratégies d'atténuation des risques pour l'Automatisation des tests.

4.3 Maintenabilité de la solution d'Automatisation des tests

TAE-4.3.1 (K2) Expliquer quels sont les facteurs qui soutiennent et affectent la maintenabilité de la solution d'Automatisation des tests.

4.1 Développement de l'Automatisation des tests

4.1.1 Appliquer des lignes directrices qui soutiennent des activités efficaces de pilotage et de déploiement de l'Automatisation des tests

Il est important de définir le périmètre de validation d'un projet pilote d'Automatisation des tests. Un projet pilote ne prend pas beaucoup de temps à mener, mais le résultat peut avoir un impact significatif sur la direction que prend le projet.

Sur la base des informations recueillies sur le SUT et des exigences sur le projet, il convient d'évaluer ce qui suit afin de mettre en place des lignes directrices pour optimiser les efforts d'Automatisation des tests :

- Le(s) langage(s) de programmation qui sera(ont) utilisé(s).
- Les outils commerciaux sur étagère/open-source appropriés.
- Les niveaux de test à couvrir.
- Les cas de test sélectionnés.
- L'approche de développement des cas de test.

Sur la base des points énumérés ci-dessus, les TAE peuvent définir une approche initiale à suivre. Sur la base des exigences, plusieurs prototypes initiaux différents peuvent être créés pour montrer les avantages et inconvénients des différentes approches. À partir de là, les TAE peuvent décider du chemin à suivre.

La définition d'un calendrier est un élément important pour respecter les délais et garantir le succès du projet pilote. Une recommandation courante est de vérifier périodiquement l'avancement du projet pilote afin d'identifier les risques éventuels et de les atténuer.

Pendant le pilote, il est également recommandé d'essayer d'intégrer la solution et le code déjà implémenté dans le CI/CD. Cela peut mettre en évidence des problèmes précoces, soit dans le SUT, soit dans la TAS, soit dans l'intégration globale des différents outils au sein de l'organisation.

Au fur et à mesure que le nombre de cas de test augmente, les TAE peuvent penser à modifier la configuration initiale du CI/CD pour exécuter les tests de différentes manières et à différents temps de réflexion.

Par ailleurs, au cours du projet pilote, il est nécessaire d'évaluer d'autres aspects non-techniques, tels que :

- Les connaissances et l'expérience des membres de l'équipe.
- La structure de l'équipe.
- Les règles de licence et d'organisation.
- Le type de plan de test et les niveaux de test ciblés à couvrir pendant l'automatisation des cas de test.

Une fois le projet pilote terminé, l'effort doit être évalué par les TAE et les Test Managers afin d'évaluer la réussite ou la défaillance et de prendre une décision appropriée.

4.2 Risques associés au développement de l'Automatisation des tests

4.2.1 Analyse des risques de déploiement et planification des stratégies d'atténuation des risques pour l'Automatisation des tests.

L'interface entre le TAF et le SUT doit être prise en compte dans le cadre de la conception architecturale. Ensuite, les outils de packaging, de logging des tests et de harnais de test peuvent être sélectionnés.

Au cours de l'implémentation du projet pilote, l'expansion et la maintenance du code d'Automatisation des tests doivent être prises en compte. Ce sont des facteurs cruciaux de la phase d'évaluation du pilote et ils peuvent sérieusement affecter la décision finale.

Différents risques liés au déploiement peuvent être identifiés à partir du pilote :

- Ouvertures de pare-feu
- Utilisation des ressources (par exemple, CPU et RAM)

Il faut se préparer aux risques liés au déploiement, tels que les problèmes de pare-feu, l'utilisation des ressources, la connexion au réseau et la fiabilité. Ces éléments ne sont pas strictement liés à l'Automatisation des tests, mais les TAEs doivent s'assurer que toutes les conditions sont réunies pour fournir des points de contrôle de qualité fiables et bénéfiques dans leur processus de développement.

L'utilisation d'appareils réels pour l'Automatisation des tests mobiles en est un exemple. Les appareils mobiles doivent être mis sous tension, disposer d'une autonomie de batterie suffisante pour fonctionner pendant le test, être connectés à un réseau et avoir accès au SUT.

Les risques techniques liés au déploiement peuvent inclure :

- Le packaging
- Le logging
- La structuration des tests
- La mise à jour

Le packaging
Le packaging doit être pris en compte car le contrôle de la version de l'Automatisation des tests est tout aussi important que pour le SUT. Le testware peut avoir besoin d'être téléchargé dans un référentiel pour être partagé au sein d'une organisation, que ce soit sur les lieux ou dans le cloud.

Logging
Le logging des tests donne la plupart des informations sur les résultats du test. Il existe plusieurs niveaux de logging des tests et tous sont utiles dans l'Automatisation des tests pour diverses raisons :

- Fatal : ce niveau est utilisé pour logger les événements d'erreur qui peuvent conduire à l'abandon de l'exécution du test.
- Erreur : ce niveau est utilisé lorsqu'une condition ou une interaction échoue et que, par conséquent, le cas de test échoue également.
- Avertissement : Ce niveau est utilisé lorsqu'une condition/action inattendue se produit mais n'interrompt pas le déroulement du cas de test.
- Info : Ce niveau est utilisé pour afficher des informations de base sur un cas de test et sur ce qui se passe pendant l'exécution du test.

- Débogage : Ce niveau est utilisé pour stocker des détails spécifiques à l'exécution qui ne sont généralement pas requis pour les logs de base, mais qui sont utiles lors de l'investigation d'une défaillance du test.
- Trace : Ce niveau est similaire à Debug mais contient encore plus d'informations.

Structuration des tests

La partie la plus importante du TAS est le harnais de test et les dispositifs de test qu'il contient, les éléments qui doivent être disponibles pour que les tests puissent être exécutés. Les dispositifs de test permettent de contrôler librement l'environnement de test et les données de test. Des préconditions et des postconditions peuvent être définies pour l'exécution des tests et les cas de test peuvent être regroupés en suites de tests de plusieurs manières. Ces aspects sont également importants à évaluer au cours d'un projet pilote. En outre, les contextes de test permettent de créer des tests automatisés qui sont répétables et atomiques.

Mise à jour

L'un des risques techniques les plus courants concerne les mises à jour automatiques sur les harnais de test (par exemple, les agents) et les changements de version sur les appareils. Ces risques peuvent être atténués par des alimentations électriques adéquates, des connexions réseau appropriées et des plans de configuration des appareils adéquats.

4.3 Maintenabilité de la solution d'Automatisation des tests

4.3.1 Expliquer quels facteurs soutiennent et affectent la maintenabilité de la solution d'Automatisation des tests

La maintenabilité est fortement influencée par les normes de programmation et les attentes des TAE les uns envers les autres.

Une règle d'or est d'essayer de suivre les principes du "clean code" de Robert C. Martin (Robert C Martin, "Clean Code : A Handbook of Agile Software Craftsmanship", 2008).

En bref, les principes du « clean code » (NDT : code propre) mettent l'accent sur les points suivants :

- Utiliser une convention de nommage commune pour les classes, les méthodes et les variables avec des noms significatifs.
- Utiliser une structure de projet logique et commune.
- Éviter le codage en dur.
- Éviter un trop grand nombre de paramètres d'entrée pour les méthodes.
- Éviter les méthodes longues et complexes.
- Utiliser le logging.
- Utiliser des canevas de conception lorsqu'ils sont utiles et requis.
- Se concentrer sur la testabilité.

Les conventions de nommage sont très utiles pour identifier la cible d'une variable donnée. Le fait d'avoir des noms de variables compréhensibles tels que "loginButton", "resetPasswordButton" aide les TAE à comprendre quel composant utiliser.

Le codage en dur consiste à intégrer des valeurs dans le logiciel sans pouvoir les modifier directement. Il peut être évité en utilisant des tests pilotés par les données, de sorte que les données de test proviennent d'une source commune qui peut être maintenue plus facilement. Le codage en dur réduit le temps de développement, mais il n'est pas recommandé de l'utiliser car les données peuvent changer fréquemment, ce qui peut prendre du temps à maintenir. Il est également conseillé d'utiliser des constantes pour les variables qui ne sont pas censées changer fréquemment. Ce faisant, il est possible de réduire les sources et les endroits qui doivent être maintenus.

L'utilisation de canevas de conception est également fortement recommandée. Les canevas de conception - tels que décrits au point 3.1.5 - permettent d'implémenter un code d'Automatisation des tests structuré et correctement maintenable, à condition que les canevas de conception soient utilisés correctement.

Pour garantir la qualité du code d'Automatisation des tests, il est recommandé d'utiliser des analyseurs statiques. Les formateurs de code tels que ceux couramment utilisés dans les IDE amélioreront la lisibilité du code d'Automatisation des tests.

Outre les principes du « clean-code », il est recommandé d'utiliser une structure et une stratégie de branches dans le contrôle de version. L'utilisation de branches différentes pour les caractéristiques, les versions et les corrections de défauts facilite la compréhension du contenu des branches.

5 Stratégies d'implémentation et de déploiement de l'Automatisation des tests – 90 minutes (K3)

Mots clés

test de contrat

Objectifs d'apprentissage pour le chapitre 5:

5.1 Intégration aux pipelines CI/CD

TAE-5.1.1 (K3) Appliquer l'Automatisation des tests à différents niveaux de test dans les pipelines

TAE-5.1.2 (K2) Expliquer la gestion de configuration pour les testware

TAE-5.1.3 (K2) Expliquer les dépendances de l'Automatisation des tests pour une infrastructure API

5.1 Intégration aux pipelines CI/CD

5.1.1 Appliquer l'Automatisation des tests à différents niveaux de test dans les pipelines

L'un des principaux avantages de l'Automatisation des tests est que les tests implémentés peuvent s'exécuter sans surveillance, ce qui en fait des candidats idéaux pour être exécutés au sein de pipelines. Cela peut se faire par le biais de pipelines CI/CD, ou du pipeline utilisé pour exécuter les tests régulièrement.

Les niveaux de test sont généralement intégrés comme suit :

- Les tests de configuration pour TAF/TAS, pendant le build, peuvent être considérés comme une sous-espèce de tests de composants. Ces tests sont exécutés pendant le build d'un projet d'Automatisation des tests (TAF/TAS) et vérifient que tous les chemins d'accès aux fichiers utilisés dans les scripts de test sont corrects, que les fichiers existent réellement et qu'ils se trouvent dans les chemins d'accès spécifiés.
- Les tests de composants font partie de l'étape de build du pipeline, car ils sont exécutés sur les composants individuels, (par exemple, les classes de bibliothèque, et les composants web). Ils agissent comme des points de contrôle de qualité pour le pipeline, et constituent donc une partie cruciale d'un pipeline d'intégration continue.
- Les tests d'intégration des composants peuvent faire partie du pipeline d'intégration continue s'il s'agit de tests de composants de bas niveau ou du SUT. Dans ce cas, ces tests et les tests de composants sont exécutés ensemble.
- Les tests système peuvent souvent être intégrés dans un pipeline de déploiement continu, où ils font office de dernier point de contrôle qualité du SUT livré.
- Les tests d'intégration entre les différents composants du système font souvent partie d'un pipeline de livraison continue en tant que points de qualité. Ces tests d'intégration du système garantissent que les composants du système développés séparément fonctionnent ensemble.

De nombreux systèmes modernes d'intégration continue font la distinction entre les phases de build et de déploiement des pipelines de livraison continue. Dans ce cas, les tests de composants et les tests d'intégration de composants font partie de la première phase de build. Lorsque cette première phase est réussie (c'est-à-dire que le build et les tests se déroulent ensemble), les composants/SUT sont déployés..

Dans le cas des tests d'intégration des systèmes, des tests système et des tests d'acceptation, il existe deux approches principales pour les intégrer dans ces pipelines :

1. Les cas de test sont exécutés dans le cadre de la phase de déploiement après le déploiement du composant. Cela peut être bénéfique, car sur la base des résultats des tests, le déploiement peut échouer et également être annulé. Toutefois, dans ce cas, si les tests doivent être réexécutés, il faut procéder à un redéploiement.
2. Les cas de test sont exécutés en tant que pipeline distinct, déclenché par la réussite du déploiement. Cela peut être avantageux s'il est prévu que différentes suites de tests et divers codes d'Automatisation des tests s'exécutent sur chaque déploiement. Dans ce cas, les tests ne constituent pas un point de contrôle de qualité. Il faut donc d'autres actions, généralement

manuelles, pour revenir sur un déploiement qui n'a pas abouti.
Dans ce cas, quelques scripts de tests automatisés simples peuvent être utilisés, en tant que contrôles de déploiement, pour s'assurer que le SUT est déployé, mais ces scripts de tests automatisés ne vérifient pas l'aptitude fonctionnelle d'une manière générale.

Les pipelines peuvent également être utilisés à d'autres fins d'Automatisation des tests, telles que :

- Exécuter périodiquement différentes suites de tests : Une suite de tests de régression peut être exécutée chaque nuit (c'est-à-dire la régression nocturne), en particulier pour les suites de tests de longue durée, de sorte que l'équipe aura une image claire de la qualité du SUT le matin.
- Exécuter des tests non fonctionnels : Soit dans le cadre d'un pipeline de déploiement continu, soit séparément, pour surveiller périodiquement certaines caractéristiques de qualité non fonctionnelles du système, telles que l'efficience des performances.

5.1.2 Expliquer la gestion de la configuration pour les testware

La gestion de la configuration fait partie intégrante de l'Automatisation des tests, car l'automatisation sera souvent exécutée sur plusieurs environnements de test et versions du SUT.

La gestion de la configuration dans l'Automatisation des tests comprend :

- La configuration de l'environnement de test.
- Les données de test.
- Les suites de tests/cas de tests.

Configuration de l'environnement de test

Chaque environnement de test utilisé dans le pipeline de développement peut avoir des configurations différentes, telles que diverses URL ou informations d'identification. La configuration de l'environnement de test est généralement stockée avec le testware. Toutefois, dans le cas d'une Automatisation des tests utilisée sur plusieurs projets ou de plusieurs TAFs pour le même projet, la configuration de l'environnement de test peut faire partie de la bibliothèque principale commune ou d'un référentiel partagé.

Données de test

Les données de test peuvent également être spécifiques à l'environnement de test ou à la version et au jeu de caractéristiques du SUT. Comme pour la configuration de l'environnement de test, les données de test sont généralement stockées dans des TAFs plus petits, mais des systèmes de gestion des données de test peuvent également être utilisés.

Suites de tests/cas de tests

Une pratique courante consiste à mettre en place différentes suites de tests pour les cas de test, en fonction de leur objectif, comme les « smoke tests » ou les tests de régression. Ces suites de tests sont souvent exécutées à des niveaux de tests distincts, en tirant parti de différents pipelines et environnements de tests.

Chaque version du SUT détermine un ensemble de caractéristiques qui comprend des cas de tests et des suites de tests qui permettent d'évaluer la qualité de la version en question. Il existe différentes options dans le testware pour gérer cela :

- Une configuration de basculement des caractéristiques peut être définie pour chaque version ou environnement de test. Il existe des cas de test et des suites de tests pour tester chaque caractéristique. La bascule de caractéristiques peut être utilisée dans le testware pour identifier les suites de tests à exécuter sur une version/un environnement de tests donné(e).
- Le testware peut également être versionné avec le SUT en utilisant la même version. De cette manière, il y a une correspondance exacte entre la version du SUT et le testware qui peut le tester. Une telle version est généralement implémentée à l'aide d'un système de gestion des livraisons utilisant des balises ou des branches**.**

5.1.3 Expliquer les dépendances de l'Automatisation des tests pour une infrastructure d'API

Lors de l'Automatisation des tests d'API, il est crucial de disposer des informations suivantes sur les dépendances pour construire une stratégie adéquate :

- Connexions API : Faciliter la compréhension de la logique métier qui peut être testée automatiquement et de la relation entre les APIs.
- Documentation API : Servir de ligne de base pour l'Automatisation des tests avec toutes les informations pertinentes (par exemple, les paramètres, les en-têtes et les types distincts de demande/réponse des objets).

Les tests automatisés intégrés de l'API peuvent être réalisés soit par les développeurs, soit par les TAE. Toutefois, avec le shift-left, il est recommandé de soutenir et de répartir les tests entre différents niveaux. Le syllabus CTFL de l'ISTQB mentionne les tests d'intégration des composants et les tests d'intégration des systèmes, qui peuvent être étendus à l'aide d'une meilleure pratique appelée "test de contrat".

Test de contrat

Le test de contrat est un type de test d'intégration vérifiant que les services peuvent communiquer entre eux et que les données partagées entre les services sont conformes à un ensemble de règles spécifiées. L'utilisation des tests de contrat permet d'assurer la compatibilité entre deux systèmes distincts (par exemple, deux micro-services) pour qu'ils communiquent l'un avec l'autre. Il va au-delà de la validation des schémas, exigeant que les deux parties parviennent à un consensus sur l'ensemble des interactions autorisées tout en prévoyant une évolution dans le temps. Il capture les interactions qui sont échangées entre chaque service et les stocke dans un contrat, qui peut ensuite être utilisé pour vérifier que les deux parties y adhèrent. L'un des principaux avantages de ce type de test est que les défauts provenant des services sous-jacents peuvent être trouvés plus tôt dans le cycle de développement durable et que la source de ces défauts peut être plus facilement identifiée.

Dans l'approche de test de contrat pilotée par le consommateur, le consommateur définit ses attentes en déterminant comment le fournisseur doit répondre à ses demandes. Dans l'approche de test de contrat pilotée par le fournisseur, ce dernier crée le contrat, qui indique comment ses services fonctionnent.

6 Reporting et métriques sur l'Automatisation des tests – 150 minutes (K4)

Mots clés

mesure, métrique, logging des tests, rapport d'avancement des tests, fin de test

Objectifs d'apprentissage pour le chapitre 6:

6.1 Collecte, analyse et reporting des données d'Automatisation des tests

TAE-6.1.1 (K3) Appliquer des méthodes de collecte de données à partir de la solution d'Automatisation des tests et du système sous test.

TAE-6.1.2 (K4) Analyser les données de la solution d'Automatisation des tests et du système sous test pour mieux comprendre les résultats.

TAE-6.1.3 (K2) Expliquer comment un rapport d'avancement des tests est construit et publié.

6.1 Collecte, analyse et reporting des données de l'Automatisation des tests

6.1.1 Appliquer les méthodes de collecte des données de la solution d'Automatisation des tests et du système sous test

Les données peuvent être collectées à partir des sources suivantes :

- Logs du SUT.
  - Interface utilisateur Web/mobile.
  - APIs.
  - Applications.
  - Serveurs web.
  - Serveurs de base de données.
- Les logs de la TAF pour fournir une piste d'audit.
- Les logs de build.
- Les logs de déploiement.
- Les logs de production pour surveiller les données en production (voir le syllabus CT-PT de l'ISTQB, section 2.3).
  - Suivi des performances en production pour effectuer une analyse des tendances.
  - Les logs des tests d'efficience des performances dans un environnement de test de performance (par exemple, tests de charge, de stress et de pic).
- Captures et enregistrements d'écrans (natifs de l'outil d'automatisation ou d'une tierce partie).

Étant donné qu'un TAS dispose nativement d'un testware automatisé, ce dernier peut être amélioré pour enregistrer des informations sur son utilisation. Les améliorations apportées au testware sous-jacent peuvent être utilisées par tous les scripts de test automatisés de niveau supérieur. Par exemple, l'amélioration du testware sous-jacent pour enregistrer l'heure de début et de fin de l'exécution du test peut s'appliquer à tous les tests.

Caractéristiques de l'Automatisation des tests qui soutiennent la mesure et la génération de rapports de test

Les langages de script de nombreux outils de test soutiennent la mesure et le reporting grâce à des fonctions qui peuvent être utilisées pour enregistrer et logger des informations avant, pendant et après l'exécution des tests individuels et des suites de tests entières.

Le reporting de test sur chacune des séries de tests doit comporter une caractéristique d'analyse permettant de prendre en compte les résultats des runs de tests précédents afin de mettre en évidence des tendances, telles que des changements dans le taux de réussite des tests.

L'Automatisation des tests nécessite généralement d'automatiser à la fois l'exécution du test et la vérification du test, cette dernière étant réalisée en comparant des éléments spécifiques des résultats réels avec les résultats attendus. Cette comparaison est réalisée au mieux par un outil de test utilisant des assertions. Le niveau d'information qui est rapporté à la suite de cette comparaison doit être pris en compte. Il est important que le statut du test soit déterminé correctement (c'est-à-dire passé ou en échec). En cas d'échec, de plus amples informations sur la cause de la défaillance seront exigées (par exemple, des captures d'écran).

Les différences entre les résultats réels et les résultats attendus d'un test ne sont pas toujours évidentes, et le soutien de l'outil peut être d'une grande aide en permettant de tester des comparaisons qui ignorent

les différences attendues, telles que les dates et les heures, tout en mettant en évidence les différences inattendues.

Logging des tests

Les logs des tests sont une source fréquemment utilisée pour analyser les défauts potentiels au sein de la TAS et du SUT. La section suivante présente des exemples de log de test, classés par TAS et SUT.

Logging de la TAS

Le contexte détermine si le TAF ou l'exécution du test est responsable du logging des informations qui doivent inclure les éléments suivants :

- Le cas de test en cours d'exécution, y compris l'heure de début et de fin.
- Le statut de l'exécution du test car, si les défaillances peuvent facilement être identifiées dans les logs des tests, le TAF devrait également disposer de ces informations et en rendre compte via un tableau de bord. Le statut de l'exécution du test peut être soit passé, soit en échec, soit une défaillance du TAF. Les statuts peuvent parfois ne pas être concluants et il est important pour une organisation de les définir de manière claire et cohérente. Une défaillance de la TAS s'applique aux situations où le défaut ne se trouve pas dans la SUT.
- Les détails de bas niveau du log du test (par exemple, logging des étapes significatives du test), y compris les informations de calendrier.
- Les informations dynamiques sur le SUT (par exemple, les fuites de mémoire) que le cas de test a pu identifier à l'aide d'outils tiers. Les résultats réels et les défaillances doivent être loggés avec la suite de tests qui a été exécutée lorsque la défaillance a été détectée.
- Dans le cas des tests de fiabilité ou des tests de stress au cours desquels de nombreux cycles de test sont effectués, un compteur doit être loggé afin de déterminer facilement combien de fois les cas de test ont été exécutés.
- Lorsque les cas de test comportent des éléments aléatoires (par exemple, des paramètres aléatoires ou des étapes de test aléatoires dans les tests de transition d'état), le numéro/les choix aléatoires doivent être loggés.
- Toutes les actions effectuées par un scénario de test doivent être loggées de manière à ce que les logs de test, ou des parties de ceux-ci, puissent être lus afin de réexécuter le test avec les mêmes étapes de test et le même calendrier. Ceci est utile pour reproduire une défaillance identifiée et pour capturer des informations supplémentaires. Les informations relatives à l'action du cas de test peuvent également être loggées par le SUT afin d'être utilisées lors de la reproduction de défaillances identifiées par le client. Si un client exécute une suite de tests, les informations du log de test sont capturées et peuvent ensuite être rejouées par l'équipe de développement lorsqu'elle teste un défaut.
- Des captures d'écran peuvent être enregistrées pendant l'exécution du test pour une utilisation ultérieure lors de l'analyse des causes racines.
- Chaque fois qu'une suite de tests déclenche une défaillance, la TAS doit s'assurer que toutes les informations nécessaires à l'analyse du défaut sont disponibles/stockées, ainsi que toute information concernant la suite de test, le cas échéant. La TAS doit sauvegarder tous les vidages de mémoire d'erreur et traces de pile associés. En outre, tous les logs des tests qui peuvent être écrasés (par exemple, les tampons cycliques sont souvent utilisés pour les logs de test sur le SUT) doivent être stockés à un endroit où ils seront disponibles pour une analyse ultérieure.

- L'utilisation de couleurs peut aider à distinguer les différents types d'informations du log de test (par exemple, les défauts en rouge et les informations sur la progression en vert).

Logging du SUT

La corrélation des résultats de l'Automatisation des tests avec les logs du SUT pour aider à identifier la cause racine des défauts dans le SUT et la TAS.

- Lorsqu'un défaut est identifié dans le SUT, toutes les informations nécessaires à l'analyse du défaut doivent être loggées, y compris les horodatages (NDT: time stamp), l'emplacement de la source du défaut et les messages d'erreur.
- Au démarrage d'un système, les informations relatives à la configuration doivent être loggées dans un fichier, comprenant, par exemple, les différentes versions du logiciel/micrologiciel, la configuration du SUT et la configuration du système d'exploitation.
- Grâce à l'Automatisation des tests, les logs des tests peuvent être facilement consultables. Une défaillance identifiée dans le log de test par la TAS devrait être facilement identifiée dans le log de test du SUT, et vice versa, avec ou sans outils supplémentaires. La synchronisation de divers logs de test avec un horodatage facilite la corrélation de ce qui s'est passé lorsqu'une défaillance est rapportée.

Intégration avec d'autres outils tiers (par exemple, feuilles de calcul, XML, documents, bases de données et outils de reporting)

Lorsque les informations issues de l'exécution des cas de test automatisés sont utilisées dans d'autres outils pour le suivi et le reporting (par exemple, la mise à jour des informations de traçabilité), il est possible de fournir les informations dans un format adapté aux outils tiers. Cela est souvent possible grâce aux fonctionnalités existantes des outils de test (par exemple, les formats d'exportation pour les rapports de test) ou en créant des rapports personnalisés qui sont produits dans un format compatible avec d'autres logiciels.

Visualisation des résultats du test

Les résultats des tests peuvent être rendus visibles à l'aide de graphiques. Envisagez d'utiliser des icônes de couleur, comme des feux tricolores, pour indiquer l'état général de l'exécution du test/de l'Automatisation des tests, afin que des décisions puissent être prises sur la base des rapports des tests. Le management est particulièrement intéressé par les résumés visuels des résultats des tests, ce qui facilite la prise de décision. S'ils ont besoin de plus d'informations, ils peuvent toujours aller plus loin dans les détails.

6.1.2 Analyser les données de la solution d'Automatisation des tests et du système sous test pour mieux comprendre les résultats de test

Après l'exécution du test, il est important d'analyser les résultats du test, d'identifier les défaillances possibles à la fois dans le SUT et dans la TAS. Pour une telle analyse, les données collectées auprès de la TAS sont primaires et les données collectées auprès du SUT sont secondaires.

- Analyser les données de l'environnement de test pour soutenir le dimensionnement approprié de l'Automatisation des tests (par exemple, dans le cloud).
    - Les regroupements et les ressources (par ex.).
    - Exécution des tests sur un seul navigateur ou plusieurs (c'est-à-dire multi-navigateurs).
- Comparer les résultats du test des exécutions précédentes.

● Déterminer comment utiliser les logs web pour surveiller l'utilisation du logiciel.

Les défaillances de l'exécution du test doivent être analysées, car il existe des problèmes potentiels :

1. Vérifier si la même défaillance s'est produite lors des exécutions du test précédentes. Il peut s'agir d'un défaut connu dans le SUT ou la TAS. Le système d'évaluation des tests peut être construit de manière à logger l'historique des résultats des cas de test, ce qui facilite l'analyse.
2. Si le défaut n'est pas connu, identifier le cas de test et ce qu'il teste. Le test peut être explicite, ou le cas de test peut être identifié dans le système de gestion des tests, sur la base de son ID loggé avec l'exécution du test.
3. Trouver à quelle étape du cas de test la défaillance s'est produite. La TAS le logge.
4. Analyser les informations du log de test sur l'état du SUT et vérifier s'il correspond aux résultats attendus en utilisant les captures d'écran, les logs API et réseau, ou tout autre log qui montre l'état du SUT.
5. Si l'état du SUT ne correspond pas à ce qui était attendu, logger un défaut dans le système de gestion des défauts. Veillez à inclure toutes les informations nécessaires sur le défaut et les logs qui justifient qu'il s'agit bien d'un défaut.

En cas de défaillance, il est possible que le résultat réel et le résultat attendu du SUT ne correspondent pas. Dans ce cas, il est très probable que le SUT contienne un défaut qui doit être corrigé, ou qu'il y ait une non-concordance invisible.

Une autre situation peut se produire si l'environnement de test n'est pas disponible pendant le run de test, ou s'il n'est que partiellement disponible. Dans ce cas, tous les cas de test peuvent échouer, soit avec le même défaut, soit si des parties du système sont en panne, avec des défaillances apparemment réelles. Pour identifier la cause racine de ces défauts, les logs du SUT peuvent être analysés, ce qui montrera s'il y avait des pannes de l'environnement de test au moment du run de test.

Si le SUT implémente des logs d'audit pour les interactions des utilisateurs (c'est-à-dire les sessions UI ou les appels API), cela aide à analyser les résultats des tests. Il y a généralement un ID unique ajouté à l'interaction avec le même ID pour chaque appel et intégration ultérieurs dans le système. De cette manière, en connaissant l'ID unique d'une requête/interaction, le comportement du système peut être observé et retracé.

Cet identifiant unique est généralement appelé identifiant de corrélation ou identifiant de trace. Il peut être loggé par la TAS pour faciliter l'analyse des résultats du test.

6.1.3 Expliquer comment un rapport d'avancement des tests est élaboré et publié

Les logs des tests fournissent des informations détaillées sur les étapes du test, les actions à entreprendre et les réponses attendues d'un cas de test et/ou d'une suite de tests. Cependant, les logs de test ne peuvent à eux seuls fournir une bonne vue d'ensemble des résultats du test. Pour cela, il est nécessaire de disposer d'une fonctionnalité de reporting des tests. Après l'exécution d'une suite de tests, un rapport d'avancement des tests concis doit être créé et publié. Un générateur de rapports peut être utilisé à cet effet.

Contenu d'un rapport d'avancement des tests

Le rapport d'avancement des tests doit contenir les résultats des tests, des informations sur le SUT et une documentation sur l'environnement de test dans lequel les tests ont été exécutés, dans un format approprié pour chacune des parties prenantes.

Il est nécessaire de savoir quels sont les tests qui ont échoué et les raisons de ces défaillances. Pour faciliter les corrections, il est important de connaître l'historique de l'exécution du test et de savoir qui a rapporté la défaillance (c'est-à-dire, en général, la personne qui l'a créé ou qui l'a mis à jour pour la dernière fois). La personne responsable doit rechercher la cause de la défaillance, rapporter le défaut qui y est lié, assurer le suivi de la correction du défaut et tester que la correction a été correctement implémentée.

Le reporting des tests est également utilisé pour diagnostiquer les éventuelles défaillances des composants du TAF.

La publication des rapports de tests

Le rapport de test doit être publié à l'intention de toutes les parties prenantes concernées. Il peut être téléchargé sur un site web, dans le cloud ou sur les lieux, envoyé à une liste de diffusion ou téléchargé vers un autre outil tel qu'un outil de gestion des tests. Cela permet de s'assurer que les rapports seront revus et analysés si des personnes sont censées les recevoir par courriel ou par le biais de messages de discussion postés par un chatbot.

Une option consiste à identifier les parties problématiques du SUT, et à conserver un historique des rapports de test, de sorte que des statistiques sur les cas de test ou les suites de tests présentant des régressions fréquentes puissent être rassemblées pour l'analyse des tendances.

Les parties prenantes auxquelles il faut rendre compte comprennent :

- Les parties prenantes du management.
  - Rôles typiques : architecte de solutions ou d'entreprise, chef de projet/de livraison, gestionnaire de programme, Test Manager ou directeur de test.
- Parties prenantes opérationnelles
  - Rôles typiques : Product Owner/manager, représentant du métier ou Analyste Métier.
- Parties prenantes techniques
  - Rôles typiques : chef d'équipe, scrum master, administrateur web, développeur/administrateur de base de données, leader de test, TAE, testeur, ou développeur.

Les rapports de tests peuvent varier en contenu ou en détail en fonction des destinataires. Alors que les parties prenantes techniques peuvent être plus intéressées par les détails de bas niveau, le management se concentrera sur les tendances, telles que le nombre de scénarios de test ajoutés depuis le dernier test run, l'évolution du ratio réussite-échec et la fiabilité de la TAS et du SUT. Les parties prenantes de l'exploitation mettent généralement l'accent sur les métriques liées à l'utilisation du produit.

Création de tableaux de bord

Les outils de reporting modernes offrent plusieurs options de reporting par le biais de tableaux de bord, de graphiques colorés, de collectes de logs détaillés et d'analyses automatisées des logs des tests. Il existe un vaste choix d'outils disponibles sur le marché.

Ces outils soutiennent l'agrégation de données à partir de sources telles que les logs de test d'exécution du pipeline, les outils de gestion de projet et les référentiels de code. La visualisation des données fournie par ces outils aide les parties prenantes à voir les tendances et à prendre des décisions en conséquence. Ces tendances peuvent inclure des regroupements de défauts, l'augmentation/la diminution de la propagation des défauts vers certains environnements de test, la dégradation des performances des SUT et la fiabilité des builds.

Analyse des logs de test par l'intelligence artificielle/le machine learning.
Ces dernières années, certains outils d'Automatisation des tests incluent ou sont basés sur des algorithmes de machine learning (ML). L'analyse automatisée de grandes quantités de données dans les logs de test aide le TAE à réduire le temps passé à localiser les défaillances, à analyser la raison des défaillances de test (est-ce un défaut dans le SUT ou dans la TAS ?) et à regrouper les défauts communs pour le reporting des tests (voir le Syllabus CT-AI de l'ISTQB).

7 Vérifier la solution d'Automatisation des tests – 135 minutes (K3)

Mots clés

analyse statique

Objectifs d'apprentissage pour le chapitre 7 :

7.1 Vérification de l'infrastructure d'Automatisation des tests

TAE-7.1.1 (K3) Planifier la vérification de l'environnement d'Automatisation des tests, y compris la configuration des outils de test.

TAE-7.1.2 (K2) Expliquer le comportement correct pour un script de test automatisé donné et/ou une suite de tests.

TAE-7.1.3 (K2) Identifier les cas où l'Automatisation des tests produit des résultats inattendus.

TAE-7.1.4 (K2) Expliquer comment l'analyse statique peut contribuer à la qualité du code d'Automatisation des tests.

7.1 Vérification de l'infrastructure d'Automatisation des tests

7.1.1 Planifier de vérifier l'environnement d'Automatisation des tests, y compris la mise en place des outils de test

Pour savoir si l'environnement d'Automatisation des tests et tous les autres composants de la TAS fonctionnent comme prévu, il faut encore procéder à des vérifications. Ces vérifications sont effectuées, par exemple, avant de commencer l'Automatisation des tests. Des mesures peuvent être prises pour vérifier les composants de l'environnement d'Automatisation des tests. Chacune d'entre elles est expliquée plus en détail ci-dessous.

Installation, paramétrage, configuration et personnalisation de l'outil de test

La TAS est constituée de nombreux composants. Chacun d'entre eux doit être pris en compte pour garantir des performances fiables et reproductibles. Les composants exécutables, les bibliothèques de fonctions correspondantes et les fichiers de données et de configuration de soutien constituent le principal élément d'une TAS. Le processus de configuration d'une TAS peut aller de l'utilisation de scripts d'installation automatisés au placement manuel des fichiers dans les dossiers correspondants. Les outils de test, à l'instar des systèmes d'exploitation et d'autres logiciels, font régulièrement l'objet de mises à jour (service packs) ou peuvent comporter des compléments optionnels ou obligatoires afin de garantir leur compatibilité avec un environnement de SUT donné.

L'installation automatisée ou la copie à partir d'un référentiel présente des avantages. Elle peut garantir que les tests sur différents SUTs ont été effectués avec la même version de la TAS et la même configuration de la TAS, le cas échéant. Les mises à jour de la TAS peuvent être effectuées par l'intermédiaire du référentiel. L'utilisation du référentiel et le processus de mise à niveau vers une nouvelle version de la TAS doivent être les mêmes que pour les outils de développement normalisés.

Répétabilité dans la configuration/le démontage de l'environnement de test

Une TAS sera implémentée sur une variété de systèmes, de serveurs et pour soutenir les pipelines CI/CD. Pour s'assurer que la TAS fonctionne correctement dans chaque environnement de test, il est nécessaire d'avoir une approche systématique pour charger et décharger la TAS de n'importe quel environnement de test donné. Cet objectif est atteint avec succès lorsque la construction et la reconstruction de la TAS n'entraînent aucune différence perceptible dans son exploitation au sein de plusieurs environnements de test et d'un environnement à l'autre. La gestion de la configuration des composants de la TAS garantit qu'une configuration donnée peut être créée de manière fiable. Une fois cette tâche accomplie, la documentation des divers composants de la TAS permettra de savoir quels aspects de la TAS peuvent être affectés ou nécessiter des modifications lorsque l'environnement de l'utilisateur final change.

Connectivité avec les systèmes/interfaces internes et externes

Une fois qu'une TAS est installée dans un environnement de SUT donné, et avant d'utiliser le SUT, un ensemble de vérifications ou de préconditions doit être administré pour s'assurer que la connectivité avec les systèmes internes, les systèmes externes et les interfaces est disponible. Par exemple, une bonne pratique consiste à se logger sur les serveurs, à lancer les outils d'Automatisation des tests, à vérifier que les outils d'Automatisation des tests peuvent accéder au SUT, à inspecter manuellement les paramètres de configuration et à s'assurer que les permissions sont correctement définies pour le logging des tests et le reporting des tests entre les systèmes. Il est essentiel d'établir des préconditions pour l'Automatisation des tests afin de s'assurer que la TAS a été installée et configurée correctement.

Test des composants du TAF

Comme tout projet de développement logiciel, les composants du TAF doivent être testés et vérifiés individuellement. Il peut s'agir de tests fonctionnels et non fonctionnels (par exemple, efficience des performances et utilisation des ressources). Par exemple, les composants qui assurent la vérification des objets sur les systèmes graphiques doivent être testés pour un large éventail de classes d'objets afin d'établir que la vérification des objets fonctionne correctement. De même, les logs et les rapports de test doivent fournir des informations précises sur l'état de l'Automatisation des tests et le comportement du SUT. Des exemples de tests non fonctionnels peuvent inclure la compréhension de la dégradation des performances du TAF, l'utilisation des ressources du système pouvant indiquer des défauts tels que des fuites de mémoire, et un manque d'interopérabilité des composants à l'intérieur et/ou à l'extérieur du TAF.

7.1.2 Expliquer le comportement correct pour un script de test automatisé donné et/ou une suite de tests

Les suites de tests automatisées doivent être testées pour vérifier leur complétude, leur cohérence et leur comportement correct. Différents types de contrôles de vérification peuvent être appliqués pour s'assurer que la suite de tests automatisés est disponible à tout moment, ou pour déterminer si elle est apte à être utilisée.

Des mesures peuvent être prises pour vérifier la suite de tests automatisés. Il s'agit notamment de :

- Vérifier la composition de la suite de tests.
- Vérifier les nouveaux tests qui se concentrent sur les nouvelles caractéristiques du TAF.
- Prendre en compte la répétabilité des tests.
- Prendre en compte le caractère intrusif des outils de tests automatisés.

Chacun de ces points est expliqué plus en détail ci-dessous.

Vérifier la composition de la suite de tests.

Vérifier la complétude (par exemple, les cas de test ont tous les résultats attendus et les données de test sont présentes) et la bonne version du TAF et du SUT.

Vérifier de nouveaux tests portant sur de nouvelles caractéristiques du TAF

La première fois qu'une nouvelle caractéristique du TAF est utilisée dans les cas de test, elle doit être vérifiée et pilotée de près pour s'assurer que la caractéristique fonctionne correctement.

Tenir compte de la répétabilité des tests

Lors de la répétition des tests, les résultats du test doivent toujours être les mêmes. Les cas de test de la suite de tests qui ne donnent pas un résultat de test fiable (par exemple, en raison d’exécutions concurrentes mal séquencées) doivent être retirés de la suite de tests automatisés active et analysés séparément pour trouver la cause racine. Dans le cas contraire, du temps sera consacré à plusieurs reprises à ces runs de test pour analyser la défaillance.

Tenir compte du caractère intrusif des outils de test automatisés

La TAS est souvent étroitement couplée au SUT. C'est une question de conception, afin d'assurer une meilleure compatibilité en ce qui concerne le niveau d'interaction. Toutefois, cette intégration étroite peut également avoir des conséquences négatives. Par exemple, lorsque la TAS est située dans l'environnement du SUT, les fonctionnalités du SUT peuvent différer de celles des tests effectués manuellement, ce qui peut avoir un impact sur les performances ainsi que sur la qualité des tests.

Un niveau d'intrusion élevé peut révéler des défaillances pendant les tests qui ne sont pas évidentes dans la production. Si cela entraîne des défaillances au niveau des tests automatisés, la confiance dans la TAS peut chuter de manière spectaculaire. Les développeurs peuvent exiger que les défaillances identifiées par l'Automatisation des tests soient reproduites manuellement, si possible, pour faciliter l'analyse.

7.1.3 Identifier où l'Automatisation des tests produit des résultats inattendus

Lorsqu'un script de test échoue ou réussit de manière inattendue, une analyse des causes racines doit être effectuée. Il faut pour cela inspecter les logs de test, les données de performance, l'installation et le démontage du script de test.

Il est également utile d'exécuter quelques tests isolés. Les défaillances intermittentes sont plus difficiles à analyser. Le défaut peut se trouver dans le cas de test, le SUT, le TAF, le matériel ou le réseau. La surveillance des ressources du système peut fournir des indices sur la cause racine. L'analyse des logs du cas de test, du SUT et du TAF peut aider à identifier la cause racine du défaut. Le débogage peut également s'avérer nécessaire. L'identification de la cause racine peut nécessiter le soutien d'un analyste de test, d'un analyste métier, d'un développeur ou d'un ingénieur système.

Vérifier si toutes les assertions sont en place. Les assertions manquantes peuvent donner lieu à des résultats de test non concluants.

7.1.4 Expliquer comment l'analyse statique peut contribuer à la qualité du code d'Automatisation des tests

L'analyse statique du code permet de constater les vulnérabilités et les défauts du code du programme. Il peut s'agir du SUT ou du TAF.

Les analyses automatisées peuvent inspecter le code afin d'atténuer les risques. Cela permet de revoir le SUT à la recherche de défauts et de s'assurer que les normes de codage sont respectées et appliquées. Cela peut également être considéré comme une technique de détection proactive des défauts, et cela joue un rôle important dans les implémentations DevSecOps (c'est-à-dire DevOps en mettant l'accent sur la sécurité). Ces analyses se produisent tôt dans le SDLC via des pipelines afin de fournir aux équipes de développement un retour d'information immédiat.

Les résultats concernant les défauts sont généralement classés comme étant de sévérité critique, élevée, moyenne ou faible, de sorte que les équipes de développement ont la possibilité de donner la priorité aux défauts qu'elles choisissent de corriger. Certains outils d'analyse statique sont également en mesure de suggérer des corrections de code pour remédier aux défauts constatés. Ils présentent aux équipes de développement une copie des lignes de code incriminées et proposent aux développeurs une solution possible à implémenter. En outre, ces outils aident les TAE en mesurant la qualité, en suggérant des zones où commenter le code, en améliorant la conception du code pour optimiser la gestion des ressources (par exemple, en utilisant des blocs « try/catch » et de meilleures structures de boucle) et en supprimant les mauvais appels à la bibliothèque.

Comme les outils d'Automatisation des tests utilisent des langages de programmation, il existe un risque qu'un code d'Automatisation des tests inadéquat soit introduit dans le SDLC. À titre d'exemple, une pratique courante lors de l'Automatisation des tests consiste à disposer d'un nom d'utilisateur et d'un mot de passe. Il est concevable qu'un TAE puisse inclure par erreur le mot de passe en clair dans un ou plusieurs scripts de test.

Les outils d'analyse statique peuvent être utiles au code d'Automatisation des tests. Ils peuvent être utilisés pour analyser le code d'Automatisation des tests à la recherche de violations de la sécurité telles qu'un mot de passe en clair dans le code. Les outils d'analyse statique soutiennent de nombreux

langages de programmation, y compris ceux utilisés par les logiciels d'Automatisation des tests. Il est donc impératif que le TAE étende les meilleures pratiques d'analyse du code pour inclure également le code d'Automatisation des tests. Même si le code d'Automatisation des tests n'est pas nécessairement déployé avec l'ensemble du logiciel, il existe clairement des vulnérabilités potentielles si le mot de passe a été découvert dans un script de test d'automatisation qui l'accompagne (voir le Syllabus CT-SEC de l'ISTQB).

8 Amélioration continue – 210 minutes (K4)

Mots clés

validation de schéma, histogramme de test

Objectifs d'apprentissage pour le chapitre 8:

8.1 Possibilités d'amélioration continue de l'Automatisation des tests

TAE-8.1.1 (K3) Découvrir les opportunités d'amélioration des cas de test par la collecte et l'analyse de données.

TAE-8.1.2 (K4) Analyser les aspects techniques d'une solution d'Automatisation des tests déployée et fournir des recommandations d'amélioration.

TAE-8.1.3 (K3) Restructurer le testware automatisé pour l'aligner sur les mises à jour du SUT.

TAE-8.1.4 (K2) Résumer les opportunités d'utilisation des outils d'Automatisation des tests.

8.1 Possibilités d'amélioration continue de l'Automatisation des tests

8.1.1 Découvrir les possibilités d'amélioration des cas de test grâce à la collecte et à l'analyse des données

La collecte et l'analyse des données peuvent être améliorées en prenant en compte différents types de données grâce aux approches décrites ci-dessous.

Histogramme de test

Un rapport visuel des données de test représenté sous la forme d'un histogramme de test fournit des domaines d'amélioration potentiels concernant les tendances des données des cas de test. Les TAE peuvent décider des domaines d'amélioration possibles car de nombreux outils de CI/CD et de reporting des tests ont la capacité de montrer différents résultats de test et leurs données de test respectives (par exemple, les logs d'exception, les messages d'erreur et les captures d'écran). L'histogramme des tests permet également aux TAE d'identifier et de sélectionner les cas de test qui sont fragiles et de les refactoriser avec des améliorations supplémentaires ou en repensant l'implémentation réelle.

Intelligence artificielle

Une autre opportunité récente est l'utilisation de l'intelligence artificielle (IA) pour soutenir les tests et l'Automatisation des tests. Par exemple, dans les cas de test de l'interface utilisateur, les données comprennent également des valeurs de localisateur d'interface utilisateur qui peuvent être traitées comme des entrées. Des outils de pointe récents permettent de détecter si un localisateur donné est modifié par rapport à celui qui est utilisé. Sur la base du ML et de la reconnaissance d'images, ils peuvent identifier les nouveaux sélecteurs et utiliser un algorithme d'auto-réparation pour corriger le cas de test et inclure les localisateurs modifiés dans le rapport de test. Cela peut accélérer les étapes de suivi telles que les changements de contrôle de version et la maintenance du code.

Validation de schéma

La validation de schéma peut être appliquée à l'analyse des données de l'API (par exemple, les propriétés dérivées des points de terminaison cibles) et à l'analyse des bases de données (par exemple, les règles de validation des champs logiciels, telles que les types de données et les plages de valeurs autorisés).

Avec la validation de schéma, la TAS est capable de vérifier si une réponse correspond à la spécification métier réelle. Ce type de contrôle peut être utilisé pour déterminer si les éléments de réponse obligatoires sont présents dans la réponse du service et si leur type d'objet correspond à celui défini dans le schéma. En cas de rupture du schéma, la solution renvoie la validation réelle qui aide les TAE à identifier la cause racine du problème.

Exemple : une API a six éléments de réponse obligatoires qui doivent être des chaînes de caractères dans la réponse. Avec les outils de validation de schéma, il n'est pas nécessaire d'écrire des assertions individuelles pour vérifier si ces types sont des chaînes et si leurs valeurs ne sont pas nulles. La validation de schéma se charge de ces vérifications, ce qui raccourcit considérablement le code d'Automatisation des tests implémenté et accroît l'efficience de la détection des défauts dans le service backend.

8.1.2 Analyser les aspects techniques d'une solution d'Automatisation des tests déployée et formuler des recommandations d'amélioration.

Outre les tâches de maintenance permanente nécessaires pour maintenir la synchronisation de la TAS avec le SUT, il existe de nombreuses possibilités d'améliorer la TAS. Ces améliorations peuvent être

apportées pour apporter toute une série de bénéfices, notamment une plus grande efficience (par exemple, en réduisant encore l'intervention manuelle), une plus grande facilité d'utilisation, des capacités supplémentaires et un meilleur soutien pour les tests. La décision d'améliorer la TAS est influencée par les caractéristiques qui ajoutent le plus de valeur à un projet.

Les domaines spécifiques d'une TAS dont l'amélioration peut être envisagée comprennent le script, l'exécution du test, la vérification, la TAA, le TAF, l'installation et le démontage, la documentation, les caractéristiques de la TAS, ainsi que les mises à jour et les mises à niveau de la TAS. Ces domaines sont décrits plus en détail ci-dessous.

L'écriture de scripts

Les techniques de script varient de l'écriture linéaire à l'approche des tests guidés par les données, puis à l'approche plus sophistiquée des tests guidés par les mots-clés, comme décrit à la section 3.1.4. Il peut être judicieux de mettre à niveau la technique de script TAS actuelle pour tous les nouveaux tests automatisés. La technique peut être adaptée à tous les tests automatisés existants, ou du moins à ceux qui nécessitent le plus d'efforts de maintenance.

Un autre domaine d'amélioration de la TAS pour les scripts de test peut se concentrer sur leur mise en œuvre. Par exemple :

- Évaluer le chevauchement des scripts de test/cas de test/étapes de test pour consolider les tests automatisés. Les cas de test contenant des séquences d'actions similaires ne devraient pas mettre en œuvre ces étapes de test plusieurs fois. Ces étapes de test doivent être transformées en une fonction et ajoutées à une bibliothèque, afin qu'elles puissent être réutilisées. Ces fonctions de bibliothèque peuvent ensuite être utilisées par différents cas de test. La maintenabilité du logiciel de test s'en trouve améliorée. Lorsque les étapes de test ne sont pas identiques mais similaires, la paramétrisation peut être nécessaire. Remarque : il s'agit d'une approche typique dans les tests pilotés par mots-clés.
- Établir un processus de reprise en cas de défaillance pour le TAS et le SUT. Lorsqu'une défaillance se produit pendant l'exécution d'une suite de tests, la TAS doit être en mesure de récupérer et de continuer avec le prochain test possible. Lorsqu'une défaillance se produit dans le SUT, la TAS doit effectuer les actions de récupération nécessaires sur le SUT (par exemple, un redémarrage du SUT) lorsque cela est faisable et pratique.
- Évaluer les mécanismes d'attente pour s'assurer que le meilleur type est utilisé. Il existe trois mécanismes d'attente courants :
    - Les attentes codées en dur (c'est-à-dire attendre un certain nombre de millisecondes) qui peuvent être à l'origine de nombreux défauts dans l'automatisation des tests, étant donné l'imprévisibilité des temps de réponse des logiciels.
    - L'attente dynamique par interrogation (par exemple, vérifier qu'un certain changement d'état ou qu'une certaine action a eu lieu) est beaucoup plus souple et efficace :
        - La TAS n'attend que le temps nécessaire, et aucun temps de test n'est perdu
        - Lorsque le processus prend plus de temps que prévu, l'interrogation attendra jusqu'à ce que la condition soit vraie. Il est recommandé d'inclure un mécanisme

de temporisation afin d'éviter que le test attende indéfiniment lorsqu'un défaut est présent.

- Un moyen encore plus efficace est de s'abonner au mécanisme d'événement du SUT. Cette méthode est beaucoup plus fiable que les deux autres options, mais le langage de script de test doit prendre en charge l'abonnement aux événements et le SUT doit proposer ces événements à la TAS. Un mécanisme de temporisation est également nécessaire, sinon le test peut attendre indéfiniment s'il y a un défaut.

Exécution des tests

Lorsqu'une suite de tests de régression automatisés n'est pas terminée parce que l'exécution des tests prend trop de temps, il peut être nécessaire de tester simultanément sur différents environnements de test quand cela est possible. Lorsque des systèmes coûteux sont utilisés pour les tests, il peut être contraignant d'effectuer tous les tests sur un seul système cible. Il peut être nécessaire de diviser la suite de tests de régression en plusieurs parties, chacune s'exécutant sur une période de temps définie (par exemple, en une seule nuit). Une analyse plus poussée de la couverture de l'automatisation des tests peut révéler des doublons. La suppression des doublons peut réduire le temps d'exécution des tests et permettre d'autres gains d'efficacité. Dans le cas de CI/CD, une bonne pratique consiste à exécuter des travaux par lots en parallèle afin d'optimiser le temps d'exécution des tests. De même, il est bon de programmer des tâches automatisées par lots pour exécuter les différents pipelines à un moment donné, par exemple tous les matins, afin de réduire les interactions manuelles et d'accélérer le processus de développement.

Vérification

Avant de créer de nouvelles fonctions de vérification, adoptez un ensemble de méthodes de vérification standard à utiliser par tous les tests automatisés. Cela évitera la ré-implémentation des actions de vérification dans plusieurs tests. Lorsque les méthodes de vérification ne sont pas identiques mais similaires, l'utilisation de la paramétrisation permet d'utiliser une fonction pour plusieurs types d'objets.

TAA

Il peut être nécessaire de modifier la TAA pour améliorer la testabilité du SUT. Ces modifications peuvent être apportées à l'architecture du SUT et/ou à la TAA de la TAS. Cela peut permettre d'améliorer considérablement l'automatisation des tests, mais peut nécessiter des changements et des investissements importants dans les SUT/TAS. Par exemple, si le SUT doit être modifié pour fournir des API pour les tests, la TAS doit également être remaniée en conséquence. L'ajout de ce type de fonctionnalités, plus tard dans le SDLC, peut s'avérer très coûteux ; il est préférable d'y penser dès le début de l'automatisation des tests et dans les premières phases du SDLC du SUT.

TAF

Il y a souvent de nouvelles versions des bibliothèques de base utilisées dans un TAF. Il s'agit parfois de mises à jour majeures, et la dernière version ne peut pas être immédiatement référencée dans la liste des dépendances du TAF, car cela casserait les tests pour de nombreuses équipes qui les utilisent. Il est donc préférable d'effectuer d'abord un pilote et une analyse d'impact. Ensuite, un plan d'adoption peut être créé. Soit toutes les équipes adoptent la nouvelle version des bibliothèques de base en même temps

en mettant à jour la dépendance dans le fichier de compilation de la couche des bibliothèques de base, soit chaque équipe décide individuellement du moment de la mise à jour dans sa couche logique métier. Finalement, une fois que toutes les équipes sont prêtes à accepter la nouvelle version des bibliothèques de base, les dépendances peuvent être mises à jour dans la couche des bibliothèques de base (voir section 3.1.3).

Initialisation et démontage (« Setup » et « Teardown »)

Les actions et les configurations qui sont répétées avant ou après chaque script ou suite de tests doivent être déplacées dans les méthodes d'initialisation ou de démontage. De cette manière, toute modification ayant un impact sur le code peut être mise à jour en un seul endroit, ce qui réduit les efforts de maintenance. Par exemple, les appels aux services web peuvent être utilisés pour remplir les conditions préalables ou postérieures des tests d'interface utilisateur (par exemple, l'enregistrement de l'utilisateur, le "nettoyage" de l'utilisateur et la configuration du profil).

Documentation

Elle couvre toutes les formes de documentation, de la documentation sur l'Automatisation des tests (par exemple, ce que fait le code d'Automatisation des tests et comment il doit être utilisé) à la documentation utilisateur pour la TAS, en passant par les rapports de test et les journaux de test produits par la TAS.

Fonctionnalités de la TAS

Ajouter des caractéristiques et des fonctions à la TAS, telles que des rapports de test détaillés, des journaux de test et l'intégration à d'autres systèmes. Seules les fonctionnalités qui seront utilisées doivent être ajoutées. L'ajout de fonctionnalités inutilisées ne fait qu'accroître la complexité et diminuer la fiabilité et la maintenabilité.

Mises à jour et mises à niveau du TAF

La mise à jour ou la mise à niveau vers de nouvelles versions du TAF peut permettre de disposer de nouvelles fonctions pouvant être utilisées par les cas de test ou de corriger des défaillances. Le risque est que la mise à jour du TAF, par la mise à niveau des outils de test existants ou l'introduction de nouveaux outils, ait un impact négatif sur les cas de test existants. Il est recommandé de tester la dernière version de l'outil de test en exécutant des exemples de tests avant de déployer la nouvelle version de l'outil de test. Les exemples de tests doivent être représentatifs des tests automatisés de différents SUT, de différents types de tests et, le cas échéant, de différents environnements de test.

8.1.3 Restructurer le logiciel de test automatisé pour l'aligner sur les mises à jour du système sous test

L'application d'un ensemble donné de modifications à un SUT existant nécessitera des mises à jour de la TAS, y compris du TAF et des bibliothèques de composants. Tout changement, aussi insignifiant soit-il, peut avoir un impact négatif important sur la fiabilité et la performance de la TAS.

Identifier les changements dans les composants de l'environnement de test

Évaluer les changements et les améliorations à apporter. Faut-il modifier le logiciel de test, les bibliothèques de fonctions personnalisées ou le système d'exploitation ? Chacun de ces éléments a un impact sur les performances de la TAS. L'objectif global est de s'assurer que les tests automatisés continuent à fonctionner de manière efficace. Les changements doivent être apportés de manière progressive, dans

l'optique d'un produit minimum viable, de sorte que l'impact sur la TAS puisse être mesuré au moyen d'une série limitée de scripts de test. Une fois que l'on a constaté qu'il n'y a pas d'effet secondaire, les changements peuvent être entièrement mis en œuvre. L'exécution d'une régression complète est la dernière étape permettant de vérifier que le changement n'a pas eu d'effet négatif sur les scripts de test automatisés. Au cours de l'exécution de ces scripts de test de régression, des défaillances peuvent être constatées. L'identification de la cause racine de ces défaillances (par exemple, à travers les rapports de test, les logs de test et l'analyse des données de test) fournira un moyen de s'assurer qu'ils ne résultent pas de l'activité d'amélioration de l'Automatisation des tests.

Accroître l'efficacité et l'efficience des bibliothèques de fonctions de base de la TAS

Au fur et à mesure qu'une TAS évolue, on découvre de nouvelles façons d'exécuter les tâches plus efficacement. Ces nouvelles techniques (par exemple, l'optimisation du code dans les fonctions et l'utilisation de nouvelles bibliothèques de système d'exploitation) doivent être intégrées dans les bibliothèques de fonctions de base utilisées par le projet actuel et les projets futurs.

Cibler plusieurs fonctions qui agissent sur le même type de contrôle pour la consolidation

Une grande partie de l'exécution d'un test automatisé consiste à interroger les contrôles dans l'interface graphique. Cette interrogation sert à fournir des informations sur un contrôle (par exemple, visible/non visible, activé/non activé, taille et dimensions, données). Grâce à ces informations, un test automatisé peut sélectionner un élément dans une liste déroulante, saisir des données dans un champ et lire une valeur dans un champ. Plusieurs fonctions peuvent agir sur les contrôles pour obtenir ces informations. Certaines fonctions sont extrêmement spécialisées, tandis que d'autres sont de nature plus générale. Par exemple, il peut exister une fonction spécifique qui ne fonctionne qu'avec les listes déroulantes. Il peut aussi y avoir une fonction qui fonctionne avec plusieurs fonctions en spécifiant une fonction comme l'un de ses paramètres. Par conséquent, un TAE peut utiliser plusieurs fonctions qui peuvent être regroupées en un nombre réduit de fonctions, ce qui permet d'obtenir les mêmes résultats et de minimiser la maintenance.

Refonte de la TAA pour tenir compte des changements dans le SUT

Tout au long du cycle de vie d'une TAS, des modifications devront être apportées pour tenir compte des changements dans le SUT. Au fur et à mesure que le SUT évolue et mûrit, la TAA sous-jacente devra également évoluer pour s'assurer que la capacité est là pour soutenir le SUT. Lors de l'extension des fonctionnalités, il convient de veiller à ce qu'elles ne soient pas mises en œuvre de manière ponctuelle, mais qu'elles soient analysées et modifiées dans le cadre de la TAA. Ainsi, lorsque de nouvelles fonctionnalités du SUT nécessiteront des scripts de test supplémentaires, des composants compatibles seront en place pour prendre en charge ces nouveaux tests automatisés.

Conventions de nommage et normalisation

Au fur et à mesure que des changements sont introduits, les conventions de nommage pour le nouveau code d'automatisation des tests et les bibliothèques de fonctions doivent être cohérentes avec les normes définies précédemment (voir section 4.3.1).

Évaluation des scripts de test existants pour la révision/élimination des SUT

Le processus de changement et d'amélioration comprend également un audit des scripts de test existants, de leur utilisation et de leur valeur continue. Par exemple, si certains tests sont complexes et longs à exécuter, il peut être plus viable et plus efficient de les décomposer en tests plus petits. En ciblant

l'élimination des tests qui ne sont pas ou peu exécutés, on réduit la complexité de la TAS et on clarifie ce qui doit être maintenu.

8.1.4 Résumer les possibilités d'utilisation des outils d'Automatisation des tests

Outre les tests proprement dits, l'automatisation des tests peut contribuer à des activités de test non spécifiques telles que :

Configuration et contrôle de l'environnement

Certains scripts de test (par exemple, la création de données de test) peuvent être exploités dans une méthode de configuration pour créer différentes données de test dans un nouvel environnement de test. Dans une situation où des utilisateurs avec plusieurs profils doivent être créés sur la base de différentes entrées de données, une équipe peut utiliser un script de test automatisé pour appeler un point de terminaison de service web qui enregistre ces utilisateurs. Les scripts de test peuvent avoir un contrôle sur la mise en place de l'infrastructure de test et peuvent être exploités dans le cadre d'un nettoyage après le processus de test. Cela permet de gagner du temps et de s'assurer que les bons utilisateurs sont présents dans chaque nouvel environnement de test. Par exemple, les différents logs de test et autres testware peuvent être retirés automatiquement d'un environnement de test, ce qui rend l'entretien et l'utilisation de l'environnement de test plus efficaces.

Vieillissement des données

L'Automatisation des tests peut être utilisée pour manipuler les données de test dans l'environnement de test. Par exemple, dans les bases de données, les champs de date peuvent être vérifiés et contrôlés pour les maintenir à jour d'une année à l'autre.

Génération de captures d'écran et de vidéos

La plupart des outils modernes d'Automatisation des tests de l'interface utilisateur ont une capacité intégrée à créer des captures d'écran ou des vidéos, sous certaines conditions, et à les stocker. Grâce à ces outils de test, les équipes peuvent soutenir le métier en créant des captures d'écran et des vidéos d'utilisation réelle pour documenter la version testée du logiciel ou à des fins de marketing.

9 Références

Normes

Les normes relatives à l'Automatisation des tests sont notamment les suivantes :

The Automatic Test Markup Language (ATML) by IEEE (Institute of Electrical and Electronics Engineers) consisting of ● IEEE Std 1671.1: Test Description ● IEEE Std 1671.2: Instrument Description ● IEEE Std 1671.3: UUT Description ● IEEE Std 1671.4: Test Configuration Description ● IEEE Std 1671.5: Test Adaptor Description ● IEEE Std 1671.6: Test Station Description ● IEEE Std 1641: Signal and Test Definition IEEE Std 1636.1: Test Results
ISO/IEC/IEEE 29119-5 (2016) Software and systems engineering – Software testing – Part 5: Keyword-Driven Testing
ISO/IEC 30130:2016 (E) Software engineering — Capabilities of software testing tools The Testing and Test Control Notation (TTCN-3) by ETSI (European Telecommunication Standards Institute) and ITU (International Telecommunication Union) consisting of ● ES 201 873-1: TTCN-3 Core Language ● ES 201 873-2: TTCN-3 Tabular Presentation Format (TFT) ● ES 201 873-3: TTCN-3 Graphical Presentation Format (GFT) ● ES 201 873-4: TTCN-3 Operational Semantics ● ES 201 873-5: TTCN-3 Runtime Interface (TRI) ● ES 201 873-6: TTCN-3 Control Interface (TCI) ● ES 201 873-7: Using ASN.1 with TTCN-3 ● ES 201 873-8: Using IDL with TTCN-3 ● ES 201 873-9: Using XML with TTCN-3 ● ES 201 873-10: TTCN-3 Documentation ● ES 202 781: Extensions: Configuration and Deployment Support ● ES 202 782: Extensions: TTCN-3 Performance and Real-Time Testing ● ES 202 784: Extensions: Advanced Parameterization ● ES 202 785: Extensions: Behaviour Types ● ES 202 786: Extensions: Support of interfaces with continuous signals ● ES 202 789: Extensions: Extended TRI
The UML Testing Profile (UTP) by OMG (Object Management Group) specifying test specification concepts for ● Test Architecture ● Test Data ● Test Behavior ● Test Logging ● Test Management

Documents de l'ISTQB®

Identifiant	Référence
ISTQB-AL-TTA	ISTQB Certified Tester, Advanced Level Syllabus, Technical Test Analyst, Version 4.0, June 2021, available from [ISTQB-Web] ISTQB Testeur certifié, syllabus de niveau Avancé, Analyste Technique de Test, version 4.0, juin 2021, disponible en français sur le site du CFTL à l'adresse suivante - Documents Associés Aux Certifications - CFTL
ISTQB-FL	ISTQB Certified Tester, Foundation Level Syllabus, Version 4.0, April 2023, available from [ISTQB-Web] ISTQB Testeur certifié, Syllabus de niveau Fondation, version 4.0, avril 2023, disponible en français sur le site du CFTL à l'adresse suivante - Documents Associés Aux Certifications - CFTL
ISTQB-MBT	ISTQB Certified Tester, Model-Based Testing Syllabus, Version 1.0, October 2015, available from [ISTQB-Web] Testeur certifié de l'ISTQB, syllabus Model-Based Testing, version 1.0, octobre 2015, disponible en français sur le site du CFTL à l'adresse suivante - Documents Associés Aux Certifications - CFTL
ISTQB-PT	ISTQB Certified Tester, Performance Testing Syllabus, December 2018, available from [ISTQB-Web] Testeur certifié de l'ISTQB, Syllabus Test de performance, décembre 2018, disponible en français sur le site du CFTL à l'adresse suivante - Documents Associés Aux Certifications - CFTL
ISTQB-SEC	ISTQB Certified Tester, Security Tester Syllabus, Version 1.0, March 2016, available from [ISTQB-Web]
ISTQB-TAS	ISTQB Certified Tester, Test Automation Strategy Syllabus, February 2024, available from [ISTQB-Web] Testeur certifié de l'ISTQB, Syllabus Automatisation des tests - Stratégie, février 2024, disponible en français sur le site du CFTL à l'adresse suivante - Documents Associés Aux Certifications - CFTL
ISTQB-Glossary	ISTQB Glossary of Terms, available online from [ISTQB-Web] Glossaire de l'ISTQB, en français également sur [ISTQB-Web]

Livres

Paul Baker, Zhen Ru Dai, Jens Grabowski, and Ina Schieferdecker, "Model-Driven Testing: Using the UML Testing Profile", Springer 2008 edition, ISBN-10: 3540725628, ISBN-13: 978-3540725626
Efriede Dustin, Thom Garrett, Bernie Gauf, "Implementing Automated Software Testing: how to save time and lower costs while raising quality", Addison-Wesley, 2009, ISBN 0-321-58051-6

Efriede Dustin, Jeff Rashka, John Paul, “Automated Software Testing: introduction, management, and performance”, Addison-Wesley, 1999, ISBN-10: 0201432870, ISBN-13: 9780201432879
Mark Fewster, Dorothy Graham, “Experiences of Test Automation: Case Studies of Software Test Automation”, Addison-Wesley, 2012
Mark Fewster, Dorothy Graham, “Software Test Automation: Effective use of test execution tools”, ACM Press Books, 1999, ISBN-10: 0201331403, ISBN-13: 9780201331400
Robert C Martin, “Clean Code: A Handbook of Agile Software Craftsmanship”, 2008, ISBN-10: 9780132350884
James D. McCaffrey, “.NET Test Automation Recipes: A Problem-Solution Approach”, APRESS, 2006 ISBN-13:978-1-59059-663-3, ISBN-10:1-59059-663-3
Daniel J. Mosley, Bruce A. Posey, “Just Enough Software Test Automation”, Prentice Hall, 2002, ISBN-10: 0130084689, ISBN-13: 9780130084682
Manikandan Sambamurthy, “Test Automation Engineering Handbook”, January 2023, ISBN: 9781804615492
Colin Willcock, Thomas Deiß, Stephan Tobies and Stefan Keil, “An Introduction to TTCN-3” Wiley, 2nd edition 2011, ISBN-10: 0470663065, ISBN-13: 978-0470663066

Articles

Robert V. Binder, Suzanne Miller, “Five Keys to Effective Agile Test Automation for Government Programs” August 24, 2017, Software Engineering Institute, Carnegie Mellon University, https://resources.sei.cmu.edu/asset_files/Webinar/2017_018_101_503516.pdf
DoD CIO, Modern Software Practices “DevSecOps Fundamentals Guidebook: Activities & Tools”, Version 2.2, May 2023, https://dodcio.defense.gov/Portals/0/Documents/Library/DevSecOpsActivitesToolsGuidebookTables.pdf?ver=_Sylg1WJB9K0Jxb2XTvzDQ%3d%3d
Péter Földházi, “Tri-Layer Testing Architecture”, https://www.pnsqc.org/docs/PROP53522057-FoldhaziDraftFinal.pdf
Thomas Pestak, William Rowell, PhD, “Automated Software Testing Practices and Pitfalls”, September 2018, https://www.afit.edu/stat/statcoe_files/Automated%20Software%20Testing%20Practices%20and%20Pitfalls%20Rev%201.pdf
Andrew Pollner, Jim Simpson, Jim Wisnowski, “Automated Software Testing Implementation Guide for Managers and Practitioners”, October 2018, https://www.afit.edu/stat/statcoe_files/0214simp%202%20AST%20IG%20for%20Managers%20and%20Practitioners.pdf
The Selenium Project, “Page object models”, https://www.selenium.dev/documentation/test_practices/encouraged/page_object_models/

10 Appendice A – Objectifs d'apprentissage/Niveau cognitif de connaissances

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

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

Niveau 2 : Comprendre (K2)

Le candidat peut sélectionner les raisons ou les explications des instructions liées au sujet, et peut résumer, comparer, classer et donner des exemples pour le concept de test.

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

Exemples	Notes
Classer les outils de test en fonction de leur objectif et des activités de test qu'ils soutiennent.	
Comparer les différents niveaux de test.	Peut être utilisé pour rechercher des similitudes, des différences ou les deux.
Différencier les tests du débogage.	Recherche de différences entre les concepts.
Faire la distinction entre les risques projet et les risques produit.	Permet de classer séparément deux concepts (ou plus).
Expliquer l'impact du contexte sur le processus de test.	
Donner des exemples de raisons pour lesquelles les tests sont nécessaires.	
Déduire la cause racine des défauts à partir d'un profil donné de défaillances.	
Résumer les activités du processus de revue des produits d'activités.	

Niveau 3 : Appliquer (K3)

Le candidat peut exécuter une procédure lorsqu'il est confronté à une tâche familière ou sélectionner la procédure correcte et l'appliquer à un contexte donné.

Verbes d'action : Appliquer, implémenter, préparer, utiliser.

Exemples	Notes
Appliquer l'analyse des valeurs limites pour dériver des cas de test à partir d'exigences données.	Doit faire référence à une procédure / une technique / un processus, etc.
Implémenter des méthodes de collecte de métriques pour soutenir les exigences techniques et de management.	
Préparer des tests de facilité d'installation pour les applications mobiles.	
Utiliser la traçabilité pour contrôler la progression des tests afin de s'assurer de leur complétude et de leur cohérence avec les objectifs du test, la stratégie de test et le plan de test.	Peut être utilisé dans un LO qui veut que le candidat soit capable d'utiliser une technique ou une procédure. Semblable à "appliquer".

Niveau 4 : Analyser (K4)

Le candidat peut séparer les informations relatives à une procédure ou à une technique en ses éléments constitutifs pour une meilleure compréhension et peut faire la distinction entre les faits et les déductions. Une application typique consiste à analyser un document, un logiciel ou la situation d'un projet et à proposer des actions appropriées pour résoudre un problème ou une tâche.

Verbes d'action : Analyser, déconstruire, tracer les grandes lignes, prioriser, sélectionner.

Exemples	Notes
Analyser la situation d'un projet donné afin de déterminer quelles techniques de test basés sur la boîte noire ou sur l'expérience doivent être appliquées pour atteindre des objectifs spécifiques.	Examinable uniquement en combinaison avec un objectif mesurable de l'analyse. Doit être de la forme "Analyser xxxx à xxxx" (ou similaire).
Classer par ordre de priorité les cas de test d'une suite de tests donnée en vue de leur exécution, en se basant sur les risques liés au produit.	
Sélectionner les niveaux de test et les types de test appropriés pour vérifier un ensemble donné d'exigences.	Nécessaire lorsque la sélection exige une analyse.

Référence

(Pour les niveaux cognitifs des objectifs d'apprentissage)

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

11 Appendice B – Matrice de traçabilité des objectifs métier avec les objectifs d'apprentissage

Cette section énumère la traçabilité entre les objectifs métier et les objectifs d'apprentissage de l'Automatisation des tests - Ingénierie.

Objectifs métier Spécialiste de l'Automatisation des tests - Ingénierie			B01	B02	B03	B04	B05	B06	B07	B08	B09	B10	B11	B12
Chapitre 1	Introduction et objectifs de l'Automatisation des tests													
1.1	Décrire l'objectif de l'Automatisation des tests													
1.1.1	Expliquer les avantages et les inconvénients de l'Automatisation des tests	2	X											
1.2	L'Automatisation des tests dans le cycle de vie du développement logiciel													
1.2.1	Expliquer comment l'Automatisation des tests est appliquée dans les différents modèles du cycle de vie du développement logiciel	2		X										
1.2.2	Sélectionner les outils d'Automatisation des tests appropriés pour un système sous test donné	2		X										
Chapitre 2	Se préparer à l'Automatisation des tests													
2.1	Besoins de configuration d'une infrastructure et d'environnements de développement													
2.1.1	Décrire les besoins de configuration d'une infrastructure permettant l'implémentation de l'Automatisation des tests	2			X									
2.1.2	Expliquer comment l'Automatisation des tests est exploitée dans différents environnements	2			X									
2.2	Processus d'évaluation pour sélectionner les bons outils et les bonnes stratégies													

2.2.1	Analyser un système sous test pour déterminer la solution d'Automatisation des tests appropriée	4				X								
2.2.2	Illustrer les constatations techniques d'une évaluation d'outil	4				X								
Objectifs métier Spécialiste de l'Automatisation des tests - Ingénierie			B01	B02	B03	B04	B05	B06	B07	B08	B09	B10	B11	B12
Chapitre 3	Architecture d'Automatisation des tests													
3.1	Conceptions utilisées dans l'Automatisation des tests													
3.1.1	Expliquer les principales capacités d'une architecture d'Automatisation des tests	2					X							
3.1.2	Expliquer comment concevoir une solution d'Automatisation des tests	2					X							
3.1.3	Appliquer la superposition de frameworks d'Automatisation des tests	3					X							
3.1.4	Appliquer différentes approches pour l'automatisation des cas de test	3					X							
3.1.5	Appliquer les principes et les canevas de conception dans l'Automatisation des tests	3					X							
Chapitre 4	Implémentation de l'Automatisation des tests													
4.1	Développement de l'Automatisation des tests													
4.1.1	Appliquer des lignes directrices qui soutiennent des activités efficaces de pilotage et de déploiement de l'Automatisation des tests	3						X						
4.2	Risques associés au développement de l'Automatisation des tests													
4.2.1	Analyser les risques liés au déploiement et planifier des stratégies d'atténuation des risques pour l'Automatisation des tests	4							X					
4.3	Maintenabilité de la solution d'Automatisation des tests													

4.3.1	Expliquer quels sont les facteurs qui soutiennent et affectent la maintenabilité de la solution d'Automatisation des tests	2								X				
Objectifs métier Spécialiste de l'Automatisation des tests - Ingénierie			B 01	B 02	B 03	B 04	B 05	B 06	B 07	B 08	B 09	B 10	B 11	B 12
Chapitre 5	Stratégies d'implémentation et de déploiement de l'Automatisation des tests.													
5.1	Intégration aux pipelines CI/CD													
5.1.1	Appliquer l'Automatisation des tests à différents niveaux de test au sein des pipelines.	3									X			
5.1.2	Expliquer la gestion de la configuration pour le testware	2									X			
5.1.3	Expliquer les dépendances de l'Automatisation des tests pour une infrastructure d'API	2									X			
Chapitre 6	Rapports et métriques sur l'Automatisation des tests													
6.1	Collecte, analyse et reporting des données d'Automatisation des tests.													
6.1.1	Appliquer des méthodes de collecte de données à partir de la solution d'Automatisation des tests et du système.	3										X		
6.1.2	Analyser les données de la solution d'Automatisation des tests et du système sous test pour mieux comprendre les résultats des tests.	4										X		
6.1.3	Expliquer comment un rapport d'avancement des tests est construit et publié	2										X		
Chapitre 7	Vérifier la solution d'Automatisation des tests													
7.1	Vérification de l'infrastructure d'Automatisation des tests													
7.1.1	Planifier la vérification de l'environnement d'Automatisation des tests, y compris la configuration des outils de test	3											X	
7.1.2	Expliquer le comportement correct pour un script de test automatisé et/ou une suite de tests donnés	2											X	

7.1.3	Identifier les cas où l'Automatisation des tests produit des résultats inattendus	2											X	
Objectifs métier Spécialiste de l'Automatisation des tests - Ingénierie			B01	B02	B03	B04	B05	B06	B07	B08	B09	B10	B11	B12
7.1.4	Expliquer comment l'analyse statique peut contribuer à la qualité du code d'Automatisation des tests.	2											X	
Chapitre 8	Amélioration continue													
8.1	Opportunités d'amélioration continue pour l'Automatisation des tests													
8.1.1	Découvrir les opportunités d'amélioration des cas de test par la collecte et l'analyse de données	3												X
8.1.2	Analyser les aspects techniques d'une solution d'Automatisation des tests déployée et fournir des recommandations d'amélioration	4												X
8.1.3	Restructurer le testware automatisé pour s'aligner sur les mises à jour du SUT	3												X
8.1.4	Résumer les opportunités d'utilisation des outils d'Automatisation des tests	2												X

12 Appendice C – Notes de version

Le syllabus 2024 ISTQB® Automatisation des tests - Ingénierie est une mise à jour et une réécriture majeure de la version 2016. Pour cette raison, il n'y a pas de notes de version détaillées par chapitre et section. Le contenu a été considérablement amélioré pour le rôle d'ingénieur en Automatisation des tests, tandis que le contenu stratégique a été déplacé vers un nouveau syllabus ISTQB® Stratégie d'Automatisation des tests 2024.

13 Appendice D – Termes spécifiques au domaine

Nom du terme	Définition
DevSecOps	Une méthodologie qui combine le développement, la sécurité et l'exploitation et qui utilise un degré élevé d'automatisation.
canevas de modèle de flux	Une vue de haut niveau du domaine de travail, de ses composants et de leurs interconnexions.
parcours de l'utilisateur	Une série d'étapes qui montrent, du point de vue de l'expérience de la personne, comment un utilisateur interagit avec un système sous test.
canevas d'objet de page	Un canevas de conception de l'automatisation des tests permettant d'améliorer la maintenance des tests et de tester moins de code en double.
contrôle de version	Un processus d'archivage et de stockage de versions spécifiques du code source.

14 Index

Tous les termes sont définis dans le glossaire de l'ISTQB®. (http://glossary.istqb.org/).

tests de l'API, 18, 26, 28, 39

développement piloté par le comportement (BDD), 23, 27

capture/rejeux, 23, 28

test de contrat, 36, 39

tests pilotés par les données, 23, 35, 55

architecture générique d'automatisation des tests, 23

test de l'interface graphique, 18, 20, 26, 28

tests pilotés par les mots-clés, 23

scripting linéaire, 23

mesure, 40, 41

métrique, 40

tests basés sur des modèles, 23

risque, 32, 33, 52

validation de schéma, 39, 53, 54

analyse statique, 20, 49, 52

scripting structuré, 23, 30

système sous test (SUT), 15, 16, 18 40

couche d'adaptabilité des tests, 23

Automatisation des tests, 14, 15, 16, 17, 18, 21, 22, 23, 24, 25, 27, 30, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 43, 44, 48, 49, 50, 51, 52, 53, 54

ingénieur en Automatisation des tests (TAE), 16

framework d'Automatisation des testss (TAF), 17, 23, 24

solution d'Automatisation des tests (TAS), 14, 16, 18, 21, 23, 32, 40, 53

contexte de test, 32

harnais de test, 23, 25, 26, 33, 34

histogramme de test, 53, 54

logging des tests, 40, 43, 44, 48

rapport d'avancement des tests, 40, 46, 67

test scripté, 23

étape de test, 23, 40, 44

testabilité, 18, 19, 28, 35

développement piloté par les tests (TDD), 23, 27

testware, 23