Testeur Certifié

Syllabus Niveau Avancé
Automatisation de test – Ingénierie

Version 2016

International Software Testing Qualifications Board

Groupe de travail Automatisation des Tests niveau expert: Bryan Bakker, Graham Bath, Armin Born, Mark Fewster, Jani Haukinen, Judy McKay, Andrew Pollner, Raluca Popescu, Ina Schieferdecker; 2016.

Historique des révisions

Version	Date	Remarques
Projet initial	28 AOUT 2015	Premier projet
Deuxième brouillon	05 NOVEMBRE 2015	Repositionnement et mappage des LOs
Troisième brouillon	17 DECEMBRE 2015	LOs revus et affinés
Projet bêta	11 JANVIER 2016	Projet édité
Bêta	18 MARS 2016	Version bêta
Sylabus 2016	21 OCTOBRE 2016	Version release AG

Table des matières

Remerciements

Ce document a été produit par une équipe du groupe de travail niveau expert ISTQB.

L'équipe remercie l'équipe de revue et tous les comités nationaux pour leurs suggestions et leurs apports.

A l'époque où le Syllabus niveau avancé pour ce module a été achevé, le groupe de travail niveau avancé – Automatisation de test - comportait les membres suivant: Bryan Bakker, Graham Bath (Chair du groupe de travail niveau expert), Armin Beer, Inga Birthe, Armin Born, Alessandro Collino, Massimo Di Carlo, Mark Fewster, Mieke Gevers, Jani Haukinen, Skule Johansen, Eli Margolin, Judy McKay (Vice chair du groupe de travail niveau expert), Kateryna Nesmyelova, Mahantesh (Monty) Pattan, Andrew Pollner (Chair de l'automatisation du test niveau expert), Raluca Popescu, Ioana Prundaru, Riccardo Rosci, Ina Schieferdecker, Gil Shekel, Chris Van Bael.

L'équipe d'auteurs pour ce syllabus: Andrew Pollner (Chair), Bryan Bakker, Armin Born, Mark Fewster, Jani Haukinen, Raluca Popescu, Ina Schieferdecker.

Les personnes suivantes ont participé à la revue, aux commentaires et au vote de ce syllabus (Par ordre alphabétique): Armin Beer, Tibor Csöndes, Massimo Di Carlo, Chen Geng, Cheryl George, Kari Kakkonen, Jen Leger, Singh Manku, Ana Paiva, Raluca Popescu, Meile Posthuma, Darshan Preet, Ioana Prundaru, Stephanie Ulrich, Erik van Veenendaal, Rahul Verma.

Ce document a été officiellement publié par l'Assemblée Générale de l'ISTQB le 21 octobre 2016.

0. Introduction à ce syllabus

0.1 Objectif de ce document

Ce syllabus pose les bases pour la qualification du test de logiciel au niveau avancé pour l'automatisation du test - Ingénierie. L'ISTQB fournit ce syllabus:

- Aux Comités membres, pour traduire dans leur langue locale et accréditer les prestataires de formations. Les Comités nationaux peuvent adapter le syllabus aux besoins de leurs langues particulières et modifier les références pour les adapter à leurs publications locales.
- Aux Comités d'examen, pour dériver des questions d'examen dans leur langue locale adaptées aux objectifs d'apprentissage pour chaque module.
- Aux prestataires de formations, pour produire des didacticiels et déterminer des méthodes d'apprentissage appropriées.
- Aux candidats à la certification, pour se préparer à l'examen (dans le cadre d'une formation ou indépendamment).
- A la communauté internationale de logiciels et d'ingénierie système, pour faire progresser la profession de test des logiciels et des systèmes, et comme base pour des livres et des articles.

L'ISTQB peut permettre à d'autres entités d'utiliser ce syllabus pour d'autres usages, à condition qu'ils demandent et obtiennent une autorisation écrite au préalable.

0.2 Portée de ce document

0.2.1 Champ d'application

Ce document décrit les tâches d'un ingénieur en automatisation de test (TAE) dans la conception, le développement et la maintenance de solutions d'automatisation de test concrètes. Il se concentre sur les concepts, les méthodes, les outils et les processus pour des tests automatisés dynamiques fonctionnels et la relation de ces tests avec la gestion des tests, la gestion de configuration, la gestion des défauts, avec les processus de développement de logiciels et l'assurance qualité.

0.2.2 Hors périmètre

Les aspects suivants sont hors périmètre pour ce syllabus Automatisation de Test - Ingénierie:

- La gestion des tests, la création automatisée de spécifications de test et la génération automatique des tests.
- Les tâches d'un gestionnaire de tests automatisés (TAM) dans la planification, la supervision et l'ajustement du développement et l'évolution de solutions de test automatisé.
- Automatisation de tests non-fonctionnels (ex, performance, sécurité)
- Automatisation de l'analyse statique (ex., analyses de vulnérabilité) et outils de test statiques)
- Enseignement de méthodes d'ingénierie logiciel et programmation (ex, quelles normes utiliser et quelles compétences avoir pour la réalisation d'une solution d'automatisation de test)
- Enseignement de technologies logicielles (ex, quels scripts techniques utiliser pour mettre en œuvre une solution d'automatisation de test)

- Sélection de produits et de services de test logiciel (ex, quels produits et services utiliser pour une solution d'automatisation de test).

.

0.3 Le testeur certifié niveau avancé en automatisation de test - ingénierie

0.3.1 Attentes

La qualification niveau avancé est destinée aux personnes ayant déjà atteint un niveau fondation dans leur carrière et qui souhaitent développer davantage leur expertise dans un domaine spécifique. Les modules proposés au niveau avancé - spécialiste - couvrent un large éventail de sujets liés aux tests.

Un ingénieur dans le domaine de l'Automatisation de Test – Ingénierie est quelqu'un qui a une vaste connaissance du test en général, et une compréhension en profondeur du domaine spécifique de l'automatisation de test. La compréhension en profondeur est définie comme une connaissance suffisante théorique et pratique de l'automatisation pour être capable d'influencer la direction que prend une organisation et/ou un projet durant la conception, le développement et la maintenance de solutions d'automatisation de test concrètes pour des tests fonctionnels.

Le document « Vue d'ensemble des modules niveaux avancés » [ISTQB-AL-Modules] décrit les compétences opérationnelles pour ce module.

0.3.2 Entrée et exigences de renouvellement

Les critères généraux d'entrée pour le Niveau Avancé sont décrits sur le site web de l'ISTQB [ISTQB-Web], section Advanced Level.

En plus de ces critères généraux d'entrée, les candidats doivent posséder la certification ISTQB au Niveau Fondation pour passer l'examen de certification d'Automatisation de Test Niveau Avancé - Ingénierie:

0.3.3 Niveaux de connaissance

Les objectifs d'apprentissage pour chaque chapitre de ce syllabus sont rassemblés au début de chaque chapitre pour une identification claire. Chaque sujet de ce syllabus sera examiné en accord avec les objectifs d'apprentissage qui lui sont assignés.

Les niveaux cognitifs assignés aux objectifs d'apprentissage (“K-levels”) sont décrit sur le site web de l'ISTQB [ISTQB-Web].

0.3.4 Examen

Tous les examens de Certification Niveau Avancé doivent être basés sur ce syllabus, ainsi que sur le syllabus Niveau Fondation [ISTQB-FL]. Les réponses aux questions d'examen peuvent requérir l'utilisation de matériels basés sur plus qu'une section de ces syllabi.

Le format de l'examen est décrit sur le site web de l'ISTQB [ISTQB-Web], section Advanced Level. Quelques informations utiles pour ceux qui passent les examens sont aussi incluses sur le site web de l'ISTQB.

0.3.5 Accréditation

Un comité membre de l'ISTQB peut accréditer les prestataires de formation dont le contenu du cours suit ce syllabus.

Le site web de l'ISTQB [ISTQB-Web], section Advanced Level, décrit les règles spécifiques qui s'appliquent aux fournisseurs de formation pour l'accréditation de leur cours.

0.4 Parties normatives versus informatives

Les parties normatives de ce syllabus sont examinables. Ce sont les:
- Objectifs d'apprentissage
- Mots clés

Le reste de ce syllabus est informatif et élaboré sur les objectifs d'apprentissage.

0.5 Niveau de détail

Le niveau de détail dans ce syllabus permet des examens et un enseignement cohérents au niveau international. Afin d'atteindre cet objectif, le syllabus se compose de:
- Objectifs d'apprentissage pour chaque domaine de connaissance, décrivant les résultats d'apprentissage cognitif et la mentalité à acquérir (ils sont normatifs.)
- Une liste d'informations à enseigner, incluant une description des concepts clés à enseigner, les sources telles que la littérature ou les normes acceptées, et les références à des sources supplémentaires si nécessaire (elles sont informatives)

Le contenu du syllabus n'est pas une description complète de l'ingénierie de l'automatisation de test; il reflète le niveau de détail à couvrir dans un cours de formation accrédité niveau avancé.

0.6 Comment ce syllabus est organisé

Il est composé de huit chapitres majeurs. Le titre en haut donne le temps pour le chapitre.
Par exemple :

3. L'architecture générique du test automatisé 270 mins.

Montre que l'enseignement du chapitre 3 est conçu pour durer 270 minutes. Les objectifs d'apprentissage spécifiques sont listés au début de chaque chapitre.

0.7 Termes, définitions et acronymes

Beaucoup de termes utilisés dans la littérature du logiciel sont utilisés de manière interchangeable. Les définitions dans ce syllabus niveau avancé sont disponibles dans le glossaire standard des termes utilisés en test de logiciel, publié par l'ISTQB [ISTQB-Glossary].

Chacun des mots clés listés au début de chaque chapitre dans ce syllabus niveau avancé y sont définis. [ISTQB-Glossary].

Les acronymes suivants sont utilisés dans ce document:

CLI	Interface de Ligne de Commande (Command Line Interface)
EMTE	Effort de Test Manuel Équivalent (Equivalent Manual Test Effort)
gTAA	Architecture générique d'Automatisation de Test (fournit un modèle pour des solutions d'automatisation de test) (generic Test Automation Architecture)
GUI	Interface Utilisateur Graphique (Graphical User Interface)
SUT	SUT, voir objet de test (System Under Test)
TAA	Architecture d'Automatisation de Test (une instanciation de gTAA pour définir l'architecture d'une TAS) (Test Automation Architecture)
TAE	Ingénieur en Automatisation de Test (la personne qui est responsable de la conception d'un TAA, son implémentation, sa maintenance et l'évolution technique de la TAS résultante) (Test Automation Engineer)
TAF	Framework d'Automatisation de Test (Environnement requis pour l'automatisation de test incluant harnais et artefacts de test tels les bibliothèques de test) (Tests Automation Framework)
TAM	Manager de l'Automatisation de Test (la personne responsable de la planification et de la supervision du développement et de l'évolution d'une TAS) (Test Automation Manager)
TAS	Solution d'Automatisation de Test (la réalisation/implémentation d'une TAA) (Test Automation Solution)
TASt	Stratégie d'Automatisation de Test (Test Automation Strategy)
UI	Interface Utilisateur (User Interface)

1. Introduction et objectifs pour l'automatisation de test - 30 mins.

Mots clés

test d'API, test de CLI, test de GUI, SUT, architecture d'automatisation de test, framework d'automatisation de test, stratégie d'automatisation de test, automatisation de l'exécution du test, scénario de test, outils de test

Objectifs d'apprentissage pour Introduction les objectifs pour l'automatisation de test

1.1 But de l'automatisation de test

ALTA-E-1.1.1 (K2) Expliquer les objectifs, avantages et limites de l'automatisation de test

1.2 Facteurs de succès dans l'automatisation de test

ALTA-E-1.2.1 (K2) Identifier les facteurs techniques de succès d'un projet d'automatisation de test

1.1 But de l'automatisation de test

Dans le test de logiciel, l'automatisation de l'exécution des tests (dénommé automatisation de test dans ce syllabus) comprend une ou plusieurs des tâches suivantes:

- Utiliser des outils logiciels spéciaux pour contrôler et configurer les conditions préalables de test
- Exécuter des tests
- Comparer les résultats réels aux résultats prévus

Un aspect clé notable est que le logiciel utilisé pour le test doit être séparé du système sous test (SUT) lui-même afin de minimiser les interférences. Il existe des exceptions, par exemple les systèmes embarqués où le logiciel de test nécessite d'être déployé dans le SUT.

L'automatisation de test est prévue pour aider à gérer de nombreux cas de tests cohérents et répétés sur différentes versions du SUT. Mais l'automatisation de test est plus qu'un mécanisme qui exécute une suite de test sans interaction humaine. Il s'agit d'un processus de conception des tests, incluant ce qui suit:

- Logiciels
- Documentation
- Cas de test
- Environnements de test
- Données

L'outil de test est nécessaire pour les activités de test ce qui inclut:

- L'Implémentation de cas de test automatisés
- La surveillance et le contrôle de l'exécution des tests
- L'interprétation, le reporting et l'enregistrement des résultats des tests

Il existe différentes approches d'automatisation de test pour tester un SUT:

1. Test par les interfaces publiques des classes, modules ou librairies du SUT (test d'API)
2. Test par l'interface utilisateur du SUT (test d'IHM ou par ligne de commande CLI)
3. Test à travers le protocole réseau

Les objectifs de l'automatisation de test incluent:

- Améliorer l'efficacité du test
- Fournir une couverture plus large
- Réduire le cout total du test
- Réaliser des tests non réalisables par un humain
- Réduire la période de test
- Augmenter la fréquence/réduire le temps requis pour les cycles de test

Les avantages de l'automatisation de test incluent:

- Plus de tests exécutés par Build
- Les tests ne pouvant pas être réalisés manuellement sont possibles (tests en temps réel, à distance, en parallèle)
- Les tests peuvent être plus complexes
- Les tests sont exécutés plus rapidement
- Les tests sont moins sujets à des erreurs opératoires
- Utilisation plus efficiente et efficace des testeurs

- Feedback plus rapide sur la qualité du logiciel
- Fiabilité des systèmes améliorée
- Qualité des tests améliorée

Les désavantages de l'automatisation des tests incluent :

- Implication de couts supplémentaires
- Investissement initial pour configurer la TAS
- Demande des technologies additionnelles
- L'équipe doit maîtriser des compétences en développement et en automatisation
- Besoin continu de maintenance de la TAS
- Peut distraire des objectifs du test (p.ex. se concentrer sur l'automatisation des cas de test au détriment de l'exécution)
- Les tests peuvent devenir plus complexes
- Des erreurs supplémentaires peuvent être introduites par l'automatisation

Les limites de l'automatisation de test incluent :

- Tous les tests manuels ne peuvent pas être automatisés
- L'automatisation peut uniquement vérifier des résultats interprétables par une machine
- L'automatisation peut uniquement vérifier des résultats obtenus qui peuvent être vérifiés par un oracle de test automatisé
- Ne remplace pas les tests exploratoires

1.2 Facteurs de succès de l'automatisation de test

Les facteurs de succès suivants s'appliquent aux projets d'automatisation de test opérationnels, par conséquent, l'accent est mis sur les influences qui impactent le succès à long terme du projet. Les facteurs influençant le succès des projets d'automatisation de test au stade pilote ne sont pas pris en compte ici.

Les principaux facteurs de succès de l'automatisation de test sont les suivants:

L'architecture d'automatisation de test (TAA)

L'architecture d'automatisation de test (TAA) est très étroitement alignée avec l'architecture d'un produit logiciel. Les exigences fonctionnelles et non fonctionnelles que l'architecture doit supporter doivent être clairement identifiées. Typiquement, ce sont les exigences les plus importantes. Souvent la TAA est conçue pour la maintenabilité, la performance et l'apprentissage. (Voir la norme ISO/IEC 25000:2014 pour en savoir plus sur ces caractéristiques et d'autres caractéristiques non fonctionnelles.) Il est utile d'impliquer des ingénieurs logiciels qui comprennent l'architecture du SUT.

Testabilité du SUT

Le SUT doit être conçu pour être testable au moyen de tests automatisés. Dans le cas des tests IHM, cela pourrait signifier que le SUT découple autant que possible l'interaction IHM et les données de l'apparence de l'interface graphique. Dans le cas de tests d'API, cela peut signifier que plus de classes,

modules ou l'interface de ligne de commande doivent être exposés en tant que public afin qu'ils puissent être testés. Les parties testables du SUT doivent être ciblées en premier. En règle générale, un facteur clé dans le succès de l'automatisation des tests réside dans la facilité d'implémenter des scripts de test automatisés. Avec cet objectif à l'esprit, et aussi pour fournir une preuve réussie de concept, l'ingénieur en automatisation de test TAE doit identifier les modules ou les composants de la SUT qui sont facilement testés au moyen de l'automatisation et commencer à partir de là.

Stratégie d'automatisation de test

Une stratégie d'automatisation de test pratique et cohérente qui aborde la maintenabilité et la cohérence du SUT.
Il n'est peut-être pas possible d'appliquer la stratégie d'automatisation de test de la même manière aux anciennes et nouvelles parties du SUT. Lors de la création de la stratégie d'automatisation, considérez les couts, les avantages et les risques de l'appliquer à différentes parties du code.
Il convient de tenir compte du test de l'interface utilisateur et de l'API avec des cas de test automatisés pour vérifier la cohérence des résultats.

Framework d'*Automatisation* des tests *(TAF)*

Un framework d'automatisation de test (TAF) facile à utiliser, bien documenté et maintenable, prend en charge une approche cohérente de l'automatisation de test.

Afin d'établir un TAF facile à utiliser et maintenable, les éléments suivants doivent être mis en œuvre:

- Mettre en place des dispositifs de reporting: les rapports de test doivent fournir des informations (passé/échec/erreur/pas exécuté/abandonné, statistique, etc.) sur la qualité du SUT. Le reporting devrait fournir une information sur la qualité aux testeurs impliqués, aux test managers, aux développeurs, aux gestionnaires de projet et à toutes les autres parties prenantes.
- Permettre un dépannage facile: en plus de l'exécution du test et de la journalisation, le TAF doit fournir un moyen facile de dépanner les tests défaillants. Le test peut échouer en raison
  - de défaillances trouvées dans le SUT
  - de défaillances trouvées dans la TAS
  - de problèmes avec les tests eux-mêmes ou avec l'environnement de test.
- Adresser de manière appropriée l'environnement de test: les outils de test dépendent de la cohérence dans l'environnement de test. Avoir un environnement de test dédié est nécessaire pour les tests automatisés. S'il n'y a aucun contrôle de l'environnement de test et des données de test, le paramétrage des tests peut ne pas répondre aux exigences pour l'exécution des tests et il est susceptible de produire des résultats d'exécution erronés.
- Documenter les cas de test automatisés: les objectifs de l'automatisation des tests doivent être clairs, par exemple, quelles parties de l'application doivent être testées, à quel degré et quels attributs doivent être testés (fonctionnels et non fonctionnels). Cela doit être clairement décrit et documenté.
- Tracer le test automatisé: le TAF doit prendre en charge le traçage pour permettre à l'ingénieur en automatisation de test de tracer des étapes individuelles des cas de test.
- Faciliter la maintenance: Idéalement, les cas de test automatisés doivent être aisément maintenus afin que la maintenance ne consomme pas une partie significative de l'effort d'automatisation des tests. En outre, l'effort de maintenance dois être proportionnel aux

changements apportés au SUT. Pour ce faire, les cas doivent être facilement analysables, modifiables et extensibles. En outre, la réutilisation du testware automatisé devrait être élevée afin de minimiser le nombre d'éléments nécessitant des modifications

- Conserver les tests automatisés à jour: lorsque des exigences nouvelles ou modifiées provoquent l'échec des tests ou de suites de test entières, ne désactivez pas les tests en échec – corrigez-les
- Planifier le déploiement: assurez-vous que les scripts de test peuvent être facilement déployés, modifiés et redéployés.
- Retirer les tests au besoin: assurez-vous que les scripts de test automatisés peuvent être facilement retirés s'ils ne sont plus utiles ou nécessaires.
- Surveiller et restaurer le SUT: en pratique, pour exécuter continuellement un cas de test ou un ensemble de cas de test, le SUT doit être surveillé en continu. Si le SUT rencontre une erreur fatale (telle qu'un plantage), le TAF doit avoir la capacité de récupérer, ignorer le cas actuel et reprendre le test avec le cas suivant.

Le code des tests automatisés peut être complexe à maintenir. Il n'est pas inhabituel d'avoir autant de code pour les tests que de code pour le SUT. C'est pourquoi il est de la plus haute importance que le code des tests soit maintenable. Cela est dû aux différents outils de test utilisés, aux différents types de vérification utilisés et aux différents artefacts du testware qui doivent être conservés (tels que les données d'entrée de test, les oracles de test, les rapports de test). Avec ces considérations de maintenance à l'esprit, et en plus des éléments importants à considérer, certaines pratiques ne devraient pas être mises en œuvre, notamment :

- Ne créez pas de code sensible à l'interface (c'est-à-dire qui serait affecté par des modifications dans l'interface graphique ou dans des parties non essentielles de l'API).
- Ne créez pas d'automatisation de test sensible aux modifications de données ou ayant une dépendance élevée à des valeurs particulières (par exemple, entrée de test en fonction d'autres sorties de test).
- Ne créez pas un environnement d'automatisation sensible au contexte (par exemple, la date et l'heure du système d'exploitation, les paramètres de localisation du système d'exploitation ou le contenu d'une autre application). Dans ce cas, il est préférable d'utiliser des bouchons de test si nécessaire afin que l'environnement puisse être contrôlé.

2. **Plus les facteurs de succès sont satisfaits, plus le projet d'automatisation de test réussira. Tous les facteurs ne sont pas requis et, dans la pratique, tous ces facteurs sont rarement respectés. Avant de commencer le projet d'automatisation des tests, il est important d'analyser les chances de succès du projet en considérant les facteurs en place et les facteurs manquants en gardant à l'esprit les risques liés à l'approche choisie ainsi que le contexte du projet. Une fois que la TAA est en place, il est important d'analyser les choses qui manquent ou celles qui nécessitent encore du travail.** Préparation pour l'automatisation de test - 165 mins.

Mots-clés
testabilité, niveau d'intrusion, pilote, bouchon, outil d'exécution de test, crochet de test

Objectifs d'apprentissage pour Préparation pour l'automatisation de test

2.1 Exigences d'automatisation de test pour le SUT et son contexte
ALTA-E-2.1.1 (K4) Analyser un SUT pour déterminer la solution d'automatisation optimale

2.2 Outil d'évaluation et processus de sélection
ALTA-E-2.2.1 (K4) Analyser les outils d'automatisation de test pour un projet donné et signalez les constatations et recommandations techniques

2.3 Conception pour la testabilité et l'automatisation
ALTA-E-2.3.1 (K2) Comprendre les méthodes de "conception pour la testabilité" et de "conception pour l'automatisation de test" applicables au SUT

2.1 Facteurs du SUT influançant l'automatisation de test

Lors de l'évaluation du contexte du SUT et de son environnement, des facteurs qui influencent l'automatisation de test doivent être identifiés pour déterminer une solution appropriée. Ceux-ci peuvent inclure les éléments suivants:

- **Interfaces du SUT**
  Les cas de test automatisés invoquent des actions sur le SUT. Pour cela, le SUT doit fournir des interfaces via lesquelles le SUT est contrôlable. Les cas de test utilisent ces interfaces. Cela peut-être fait via les contrôles IHM, mais aussi via des interfaces logicielles de niveau inférieur. En outre, certains cas de test peuvent être capables d'interfacer au niveau communication (par exemple, à l'aide de TCP/IP, USB, ou d'interfaces de messages propriétaires).

  La décomposition du SUT permet à l'automatisation de test de s'interfacer avec le SUT à différents niveaux. Il est possible d'automatiser les tests à un niveau spécifique (par exemple, niveaux composant et système), mais uniquement lorsque le SUT prend cela en charge de manière adéquate. Au niveau composant, il peut ne pas y avoir d'interface utilisateur pouvant être utilisée pour le test. Dans ce cas, des interfaces logicielles différentes, possiblement personnalisées, (aussi appelée crochet (« hook ») de test) doivent être disponibles.

- **Logiciel tiers**
  Souvent le SUT ne se compose pas seulement d'un logiciel écrit en interne mais peut aussi inclure des logiciels fournis par des tiers. Dans certains contextes, ces logiciels tiers peuvent nécessiter des tests, et si l'automatisation de test est justifiée, cela peut nécessiter une solution d'automatisation de test différente, par exemple l'utilisation d'une API.

- **Sélection des tests à automatiser**
  Tout ne peut peut-être automatisé dans le SUT dans le temps disponible. Des choix sont à faire. Il faudrait se focaliser sur les parties du SUT où le risque est élevé (forte probabilité de défauts, et fort impact des défauts sur le SUT). Le test basé sur les risques peut être utilisé pour identifier les zones à haut risque où les efforts d'automatisation de test devraient être concentrés.

- **Niveau d'intrusion**
  Différentes approches d'automatisation de test (utilisant différents outils) ont des niveaux d'intrusion différents. Plus il y a d'adaptations requises sur le SUT, plus le niveau d'intrusion est élevé. Utiliser des interfaces logicielles dédiées requiert un haut niveau d'intrusion tandis que l'utilisation des éléments IHM existants a un niveau d'intrusion plus faible. Utiliser les éléments matériels du SUT (tels que les claviers, les commutateurs manuels, les écrans tactiles, les interfaces de communication) a un niveau d'intrusion encore plus bas.

  Le problème avec les hauts niveaux d'intrusion est le risque de fausses alarmes. La TAS peut montrer des erreurs qui peuvent être dues au niveau d'intrusion imposé par les tests, mais celles-ci ne sont pas susceptibles de se produire quand le système logiciel va être utilisé dans un environnement réel en direct. De manière générale cependant, le test avec un haut niveau d'intrusion est une solution plus simple d'approche d'automatisation de test.

- **Différentes architectures de SUT**

Différentes architectures de SUT peuvent requérir différentes solutions d'automatisation de test. Une approche différente est nécessaire pour un SUT écrit en C++ utilisant la technologie COM et pour un SUT écrit en Python. Il peut être possible pour ces différentes architectures d'être gérées par la même TASt, mais cela nécessite une TASt hybride qui ait la possibilité de les supporter.

- **Taille et complexité du SUT**
  Il est aussi important de considérer la taille et la complexité du SUT actuel et des plans pour son futur développement. Pour un simple et petit SUT, une approche d'automatisation de test complexe et ultra-flexible peut ne pas être justifiée. Une approche simple peut être mieux adaptée. A l'inverse, il peut ne pas être sage d'implémenter une petite et simple approche pour un SUT très vaste et complexe. Parfois cependant, il convient de commencer petit et simple même pour un SUT complexe. Mais ce doit être une approche temporaire (voir le chapitre 3 pour plus de détails.)

Plusieurs facteurs décrits ici sont connus (par exemple, la taille et la complexité, les interfaces logicielles disponibles) quand le SUT est déjà disponible, mais la plupart du temps le développement de l'automatisation de test doit démarrer avant que le SUT ne soit disponible. Quand cela arrive, plusieurs choses doivent être estimées, ou le TAE peut spécifier les interfaces logicielles qui sont nécessaires. (voir la section 2.3 pour plus de détails.)

Même lorsque le SUT n'existe pas encore, la planification de l'automatisation de test peut démarrer. Par exemple:

- Quand les exigences (fonctionnelles ou non fonctionnelles) sont connues, les candidats à l'automatisation peuvent-être sélectionnés à partir de ces exigences avec indentification des moyens pour les tester. La planification pour l'automatisation peut démarrer pour ces candidats, incluant l'identification des exigences d'automatisation et la détermination de la stratégie.
- Quand l'architecture et la conception technique sont en cours d'élaboration, la conception des interfaces logicielles pour supporter les tests peut-être entreprise.

2.2 Évaluation et sélection d'outils

La responsabilité principale de la sélection d'outils et du processus d'évaluation incombe au TAM (Test Automation Manager). Cependant le TAE sera impliqué dans la fourniture d'informations au TAM et dans la réalisation de nombreuses activités d'évaluation et de sélection. Le concept de l'évaluation de l'outil et du processus de sélection a été introduit au Niveau Fondation et plus de détails de ce processus sont décrits dans le Niveau Avancé – Syllabus Manager de Test [ISTQB-AL-TM].

Le TAE sera impliqué tout au long de l'évaluation des outils et du processus de sélection mais va avoir des contributions particulières pour effectuer les activités suivantes :

- Évaluation de la maturité organisationnelle et identification des opportunités pour le support d'outils de test
- Évaluation des objectifs appropriés pour le support d'outils de test
- Identification et collecte d'information sur les outils qui pourraient convenir
- Analyse des informations sur ces outils par rapport aux objectifs et contraintes du projet
- Estimation du rapport cout/bénéfices basé sur une analyse solide de rentabilité
- Émission de recommandations sur l'outil approprié
- Identification de la compatibilité de l'outil avec les composants du SUT

Les outils d'automatisation d'exécution de tests fonctionnels ne peuvent fréquemment pas satisfaire toutes les exigences ou les situations rencontrées par un projet d'automatisation. Voici une série d'exemples de ces types de problèmes (mais ce n'est certainement pas une liste complète) :

Résultats	Exemples	Solutions possibles
L'interface outils ne fonctionne pas avec d'autres outils qui sont déjà en place	• L'outil de gestion de test a été mise à jour et l'interface de connexion a changé • L'information du support d'avant-vente était fausse et toutes les données ne peuvent être transférées dans l'outil de reporting	• Faire attention aux notes de versions avant les mises à jour, et pour les grandes migrations, tester avant de migrer vers la production • Essayer d'obtenir une démonstration sur site de l'outil qui utilise le SUT réel • Rechercher le support des vendeurs et/ou des forums de communauté d'utilisateurs
Certaines dépendances de SUT sont remplacées par d'autres non prises en charge par l'outil de test	• Le département de développement est passé à la dernière version de Java	• Synchroniser les mises à jour de développement/environnement de test avec les outils d'automatisation de test
L'objet dans l'IHM ne peut pas être capturé	• L'objet est visible mais l'outil d'automatisation de test ne peut pas l'analyser	• Essayer d'utiliser uniquement des technologies bien-connues ou des objets en développement • Faire un projet pilote avant d'acheter un outil d'automatisation de test • Demander aux développeurs de définir des normes pour les objets
L'outil semble très compliqué	• L'outil à de nombreuses fonctionnalités mais seule une partie sera utilisée	• Essayer de trouver un moyen de limiter l'ensemble des fonctionnalités, par exemple en retirant de la barre d'outil les fonctionnalités inutiles • Acheter une autre licence plus appropriée. • Essayer de trouver des outils alternatifs qui sont plus focalisés sur la fonctionnalité requise.

Résultats	Exemples	Solutions possibles
Conflits avec d'autres systèmes	• Après l'installation d'autres logiciels, l'outil d'automatisation de test ne fonctionne plus ou vice versa	• Lire les notes de version ou les exigences techniques avant l'installation. • Obtenir la confirmation auprès du fournisseur qu'il n'y aura aucun impact sur les autres outils. • Questionner les forums de communauté d'utilisateurs.
Impact sur le SUT	• Pendant/après l'utilisation de l'outil de test, le SUT réagit différemment (par exemple, temps de réponse plus long)	• Utiliser un outil qui ne nécessitera pas de changer le SUT (par exemple, installation de bibliothèques, etc.)
Accès au code	• L'outil d'automatisation de test va changer des parties du code source	• Utiliser un outil qui ne nécessitera pas de changer le code source (par exemple, installation de bibliothèques, etc.)
Ressources limitées (principalement dans les environnements intégrés)	• L'environnement de test a des ressources disponibles limitées ou est à court de ressources (par exemple, mémoire)	• Lire les notes de version et discuter de l'environnement avec le fournisseur de l'outil pour avoir la confirmation que cela ne va pas conduire à des problèmes. • Questionner les forums de communauté d'utilisateurs.
Mises à jour	• La mise à jour ne va pas migrer toutes les données ou corrompt les données • La mise à jour nécessite un environnement diffèrent (meilleur)	• Tester la mise à jour sur l'environnement de test et obtenir la confirmation du fournisseur que la migration va fonctionner • Lire les pré requis à la mise à jour et décider si la mise à jour vaut la peine • Solliciter l'assistance des forums de communautés d'utilisateurs
Sécurité	• L'outil d'automatisation de test requiert des informations qui ne sont pas disponibles à l'ingénieur d'automatisation de test	• L'ingénieur d'automatisation doit obtenir un accès
Incompatibilité entre différents environnements et plateformes	• L'automatisation de test ne fonctionne pas sur tous les environnements/plateformes	• Mettre en place des tests automatisés afin de maximiser l'indépendance de l'outil ainsi que pour minimiser le cout d'utilisation d'outils multiples.

2.3 Conception pour testabilité et automatisation

La testabilité du SUT (disponibilité des interfaces logicielles qui supportent le test, par exemple, permettre le contrôle et l'observation du SUT) devrait être conçue et mise en place en parallèle de la conception et de l'implémentation des autres fonctions du SUT. Cela peut être fait par l'architecte logiciel (Comme la testabilité est juste une des exigences non fonctionnelles du système), mais souvent cela est réalisé par un TAE ou avec sa participation.

La conception pour testabilité comprend plusieurs parties:

- Observabilité: Le SUT doit fournir des interfaces qui donnent un aperçu du système. Les cas de test peuvent utiliser ces interfaces pour vérifier, par exemple, si le comportement attendu correspond au comportement obtenu.
- Contrôle(abilité): Le SUT doit fournir des interfaces pouvant être utilisées pour effectuer des actions sur le SUT. Cela peut-être des éléments d'IHM, des appels de fonction, des éléments de communication (par exemple, TCP/IP ou protocole USB), des signaux électroniques (pour les commutateurs physiques), etc.
- Architecture clairement définie: Le troisième élément important pour concevoir la testabilité est une architecture qui fournit des interfaces claires et compréhensibles offrant contrôle et visibilité à tous les niveaux de test.

Le TAE considère les façons dont on peut tester le SUT, y compris le test automatisé, d'une manière efficace (tester la bonne zone et trouver les bugs critiques) et efficiente (sans faire trop d'effort, par exemple, automatisation). Chaque fois que des interfaces logicielles spécifiques sont nécessaires, elles doivent être spécifiées par le TAE et implémentées par le développeur. Il est important de définir la testabilité et, si nécessaire, les interfaces logicielles à ajouter plus tôt dans le projet, afin que le travail pour les développeurs puisse être planifié et budgétisé.

Certains exemples d'interfaces logicielles qui supportent le test incluent:

- La capacité puissante de scripting des tableurs (feuilles de calcul) modernes
- Utiliser des pilotes ou des bouchons pour simuler une partie logicielle et/ou matérielle (par exemple, une transaction financière électronique, un service logiciel, un serveur dédié, un tableau électronique, une partie mécanique) qui ne sont pas encore disponibles ou sont trop chers à l'achat. Ils permettent le test du logiciel en l'absence de matériel spécifique.
  Des interfaces logicielles (ou des bouchons et pilotes) peuvent-être utilisées pour tester les conditions d'erreur. Considérons un dispositif avec un disque dur interne (HDD). Le logiciel contrôlant ce HDD (appelé pilote) devrait être testé pour les pannes ou l'usure du HDD. Faire cela en attendant qu'un HDD devienne défectueux n'est pas très efficace (ou fiable) Mettre en place des interfaces logicielles qui simulent un HDD défectueux ou lent, permet de vérifier que le pilote logiciel fonctionne correctement (par exemple, fournit un message d'erreur, fait de nouvelles tentatives).
- Des interfaces logicielles alternatives peuvent être utilisées pour tester un SUT lorsque aucune interface utilisateur n'est disponible (et cela est souvent considéré comme étant une meilleure approche dans tous les cas). Des logiciels embarqués dans des systèmes techniques nécessitent souvent de surveiller la température dans l'appareil et de déclencher une fonction de refroidissement lorsque la température atteint un certain niveau. Cela peut être testé sans le matériel en utilisant l'interface logicielle pour spécifier la température.

- Le test de transition d'état est utilisé pour évaluer le comportement des états du SUT. Un moyen de vérifier si le SUT est dans un état correct est de l'interroger via un logiciel spécifique conçu pour cette application (bien que cela inclut un risque, voir Niveau d'Intrusion dans la section 2.1).

La conception pour l'automatisation devrait prendre en compte que :

- La compatibilité avec les outils de test existants doit être établie dès le début.
- La question de la compatibilité des outils de test est essentielle en ce qu'elle peut avoir un impact sur la capacité d'automatiser des tests de fonctionnalités importantes (par exemple, l'incompatibilité avec un contrôle de grille qui empêche tous les tests utilisant ce contrôle).
- Les solutions peuvent nécessiter le développement de code de programme et des appels aux API

Concevoir avec un objectif de testabilité est primordial pour une bonne approche d'automatisation de test, mais cela peut aussi être utile pour l'exécution de test manuelle.

3. L'architecture d'automatisation de test générique - 270 mins.

Mots-clés

Capture/rejeu, test axé sur les données , architecture d'automatisation de test générique, test axé sur les mots-clés, script linéaire, test basé sur un modèle, script axé sur le processus, script structuré, test en couche d'adaptation, architecture d'automatisation de test, framework d'automatisation de test, solution d'automatisation de test, couche de définition de test, couche d'exécution de test, couche de génération de test

Objectifs d'apprentissage pour l'architecture d'automatisation de test générique

3.1 Introduction à la gTAA

ALTA-E-3.1.1 (K2) Expliquer la structure de la gTAA

3.2 Conception d'une TAA

ALTA-E-3.2.1 (K4) Concevoir une TAA appropriée pour un projet donné
ALTA-E-3.2.2 (K2) Expliquer le rôle joué par les couches dans une TAA
ALTA-E-3.2.3 (K2) Comprendre des considérations de conception pour une TAA
ALTA-E-3.2.4 (K4) Analyser les facteurs de l'implémentation, de l'utilisation, et de la maintenance (exigences) pour un programme de TAA donné

3.3 Développement d'une TAS

ALTA-E-3.3.1 (K3) Appliquer des composantes de la TAA générique (gTAA) pour construire une TAA

ALTA-E-3.3.2 (K2) Expliquer les facteurs à considérer lors de l'identification des composants à réutiliser

3.1 Introduction à la gTAA

Un ingénieur d'automatisation de test niveau avancé (TAE) a un rôle de concevoir, de développer, et de maintenir des solutions d'automatisation de test (TAS). Comme chaque solution est développée, des tâches similaires doivent être effectuées, des réponses apportées à des questions similaires, et des problèmes similaires doivent être étudiés et triés. Ces concepts récurrents, étapes, et approches de test fonctionnel automatisé deviennent les bases de l'architecture d'automatisation de test, appelée gTAA.

La gTAA représente les couches, composants et interfaces de la TAA générique, qui sont ensuite définies dans la TAA pour une TAS particulière. Cela permet à une approche structurée et modulaire de construire une solution d'automatisation de test en:

- Définissant l'espace de concept, couches, services et interfaces d'une TAS pour permettre la réalisation de TAS par des composants internes et de tierce partie
- Apportant de la simplification par le développement efficace et efficient des composants d'automatisation de test
- Réutilisant les composants d'automatisation de test pour des TAS différentes ou en évolution, pour des gammes et familles de produits logiciels et au travers de technologies et d'outils logiciels
- Facilitant la maintenance et l'évolution de la TAS
- Définissant les caractéristiques essentielles pour un utilisateur de la TAS

Une TAS comporte l'environnement de test ainsi que ses artefacts, et des suites de test (une suite de test inclut les données de test). Un framework d'automatisation des tests (TAF) peut être utilisé pour réaliser une TAS. Il constitue le support pour la réalisation de l'environnement de test et fournit les outils, harnais de test ou bibliothèques.

Il est recommandé que la TAA de la TAS soit conforme avec les principes suivants qui supportent un développement, une évolution et une maintenance facile de la TAS:

- Responsabilité unique: Chaque composant de la TAS doit avoir une seule responsabilité, et cette responsabilité doit être entièrement encapsulée dans le composant. En d'autres termes, chaque composant de la TAS devrait être en charge d'une chose et seulement une, par exemple, génération de mots-clés ou de données, création de scénario de test, exécution de cas de test, chargement des résultats, génération de rapports d'exécution.
- Extension (voir par exemple le principe ouvert/fermé par B. Myer): Chaque composant de la TAS doit pouvoir être étendu mais pas être modifiable. Ce principe signifie qu'il devrait être possible de modifier ou d'enrichir le comportement du composant sans casser la fonctionnalité compatible en arrière.
- Remplacement (voir par exemple le principe de substitution de B. Liskov): Chaque composant de la TAS doit être remplaçable sans affecter le comportement global de la TAS. Le composant peut être remplacé par un ou plusieurs composants de remplacement mais son comportement doit être le même.
- Principe de ségrégation de composant (voir par exemple le principe des interfaces de ségrégation de R.C.Martin): Il est préférable d'avoir des composants spécifiques plutôt qu'un composant général et polyvalent. Cela rend leur substitution et maintenance plus faciles en éliminant les dépendances inutiles.
- Inversion de dépendance: Le composant d'une TAS doit dépendre des abstractions plutôt que des détails de bas niveau. En d'autres termes, les composants ne devraient pas dépendre de scenarios de test automatisé spécifiques.

Typiquement, une TAS basée sur une gTAA va être implémentée par un ensemble d'outils, leurs plugins et/ou composants. Il est important de noter que la gTAA n'est pas liée à des fournisseurs : elle ne prédéfinit pas une méthode concrète, une technologie ou des outils pour la réalisation de la TAS. La gTAA peut être implémentée avec n'importe quelle approche d'ingénierie logicielle, par exemple, structurée, orientée objet, orientée services, pilotée par les modèles, ainsi qu'avec n'importe quelle technologie logicielle et outils. En fait, une TAS est souvent implémentée en utilisant des outils dits sur étagère, mais nécessitera typiquement des ajouts et/ou des adaptations de SUT additionnels spécifiques.

D'autres lignes directrices et modèles de référence relatifs aux TASs sont des normes d'ingénierie logicielle pour le SDLC sélectionné (cycle de développement du logiciel), les technologies de programmation, les normes de formatage, etc. Il n'est pas dans le cadre de ce programme d'enseignement du génie logiciel en général, cependant, un TAE est censé avoir des compétences, l'expérience et l'expertise en génie logiciel.

En outre, un TAE doit être au courant des normes de codage et de documentation dans l'industrie et des bonnes pratiques pour les utiliser pendant le développement de la TAS. Ces normes sont typiquement spécifiques à un domaine, les normes populaires incluent:

- MISRA pour C or C++
- Normes de codage JSF pour C++
- Règles AUTOSAR pour MathWorks Matlab/Simulink®

3.1.1 Vue d'ensemble de la gTAA

La gTAA est structurée en couches horizontales comme suit:

- Génération de test
- Définition de test
- Exécution de test
- Adaptation de test

La gTAA (voir figure 1: L'architecture d'automatisation de test générique) englobe ce qui suit:

- La couche de génération de test qui prend en charge la conception manuelle ou automatisée des cas de test. Elle fournit les moyens pour concevoir les cas de test.
- La couche de définition de test qui prend en charge la définition (et l'implémentation) de suites de test et/ou de cas de test. Cela isole la définition de test, du SUT et/ou des technologies et outils système de test. Elle fournit les moyens de définir des tests de haut-niveau et des tests de bas niveau, qui sont traités dans les données de test, les cas de test, et les composants de la librairie de test ou la combinaison de ces derniers.
- La couche d'exécution de test prend en charge l'exécution des cas de test et l'enregistrement des tests. Elle fournit un outil d'exécution de test pour exécuter les tests sélectionnés automatiquement et des composants d'enregistrement et de reporting.
- La couche d'adaptation de test fournit le code nécessaire pour adapter les différents composants ou interfaces au SUT. Elle fournit différents adaptateurs pour se connecter au SUT via les APIs, protocoles, services, et autres.
- Il dispose également d'interfaces pour la gestion de projet, la gestion de la configuration et la gestion des tests en rapport avec l'automatisation des tests. Par exemple, l'interface entre la gestion des tests et la couche d'adaptation de test porte sur la sélection et la configuration des adaptateurs appropriés au regard de la configuration de test choisie.

L'interface entre la gTAA et ses composants est typiquement spécifique et, par conséquent, n'est pas développée davantage ici.

Il est important de comprendre que ces couches peuvent être présentes ou absentes dans toute TAS donnée. Par exemple :

- Si l'exécution de test doit être automatisée, l'exécution de test et la couche d'adaptation de test doivent être utilisées. Elles n'ont pas besoin d'être séparées et peuvent être utilisées ensemble, par exemple, dans les frameworks de tests unitaires.
- Si la définition de test doit être automatisée, la couche de définition de test est requise.
- Si la génération de test doit être automatisée, la couche de génération de test est requise.

Le plus souvent, on commencerait avec l'implémentation d'une TAS du bas vers le haut, mais d'autres approches telle que la génération automatique de tests pour des tests manuels peut aussi être utile. En général il est conseillé de mettre en œuvre la TAS par étapes progressives (par exemple dans des sprints) afin d'utiliser la TAS aussi tôt que possible et de prouver sa valeur ajoutée. De même, les preuves de concept sont recommandées dans le cadre d'un projet d'automatisation de test.

Tout projet d'automatisation de test doit être compris, installé et géré comme un projet logiciel et exige une gestion du projet dédiée. La gestion du projet pour le développement du TAF (c.-à-d. le soutien à l'automatisation des tests pour toute une entreprise, les familles de produits ou les gammes de produits) peut être séparée de la gestion du projet pour la TAS (c.-à-d. l'automatisation des tests pour un produit concret).

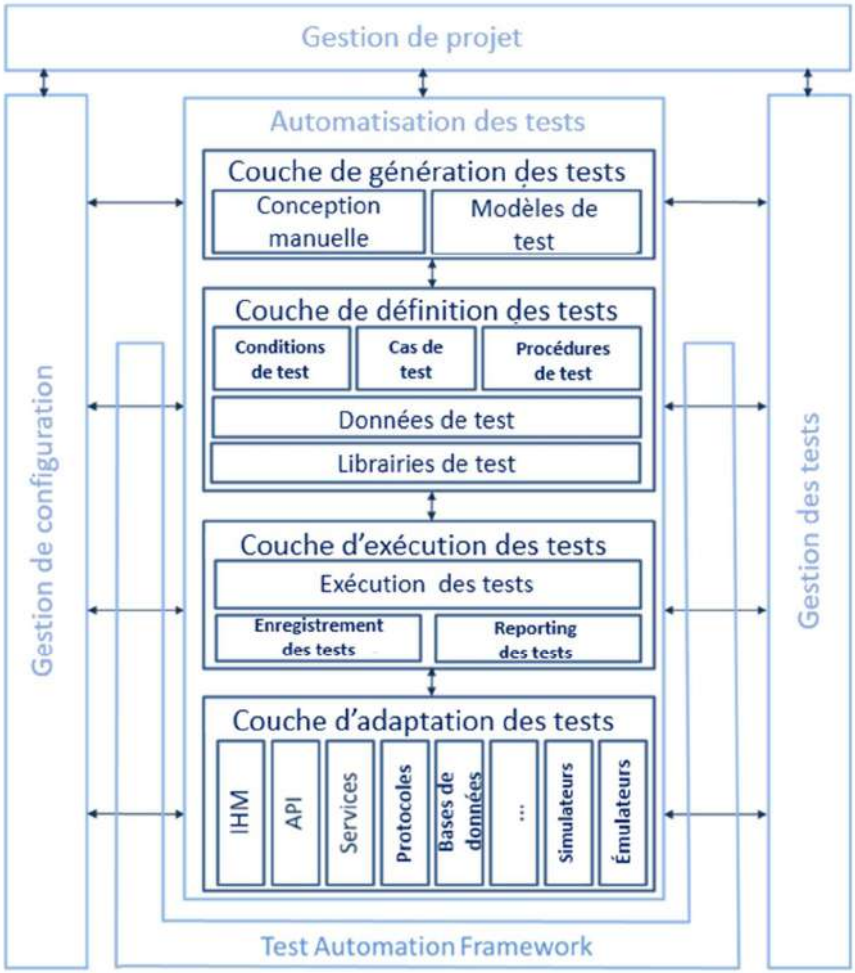

Figure 1: L'architecture d'automatisation de test générique

3.1.2 Couche de génération de test

La couche de génération de test apporte un support outillé pour ce qui suit:

- Conception manuelle de cas de test
- Développement, capture ou dérivation des données de test
- Génération automatique de cas de test depuis les modèles qui définissent le SUT et/ou son environnement (c.à.d. test basé sur les modèles automatisé)

Les composants dans cette couche sont utilisés pour :

- Éditer et naviguer dans les structures des suites de test
- Relier les cas de test aux objectifs de test ou aux exigences du SUT
- Documenter la conception du test

Pour la génération de tests automatisés les fonctionnalités suivantes peuvent également être incluses:

- Capacité à modéliser le SUT, son environnement et/ou le système de test
- Capacité à définir des directives de test et à configurer/paramétrer des algorithmes de génération de test
- Capacité à tracer à l'envers les tests générés vers le modèle (éléments)

3.1.3 Couche de définition de test

La couche de définition de test apporte un support outillé pour ce qui suit :

- Spécifier les cas de test (de haut et/ou bas niveau)
- Définir les données de test pour les cas de test de bas niveau
- Spécifier les procédures de test pour un (une suite de) cas de test
- Définir les scripts de test pour l'exécution des cas de test
- Fournir l'accès aux librairies de test selon les besoins (par exemple dans les approches axées sur les mots-clés)

Les composants dans cette couche sont utilisés pour:

- Partitionner/restreindre, paramétrer ou instancier les données de test
- Spécifier les séquences de test ou les comportements de test à part entière (incluant les instructions et expressions de contrôle), pour les paramétrer et/ou les grouper
- Documenter les données de test et/ou les cas de test

3.1.4 Couche d'exécution de test

La couche d'exécution de test se compose de support d'outils pour ce qui suit:

- Exécution automatique de cas de test
- Enregistrement d'exécution de cas de test et de leurs résultats
- Reporting des résultats

La couche d'exécution de test peut être constituée de composants qui fournissent les fonctionnalités suivantes:

- Installation et désinstallation du SUT pour l'exécution de test
- Installation et désinstallation des suites de test (c.à.d., Suite de test incluant des données de test)
- Configuration et paramétrage de l'installation de test
- Interprétation des données de test et des cas de test et transformation de ces derniers en scripts exécutables.
- Instrumentation du système de test et/ou du SUT pour (filtrer) enregistrer l'exécution de test
- Analyse des réponses du SUT pendant l'exécution de test pour orienter l'exécution de test ultérieure
- Validation des réponses du SUT (comparaison des résultats attendus et obtenus) pour les résultats de l'exécution de cas de tests automatisés
- Contrôle de l'exécution des tests automatisés au cours du temps

3.1.5 Couche d'adaptation de test

La couche d'adaptation de test propose un support outillé pour ce qui suit:

- Contrôler le harnais de test
- Interagir avec le SUT
- Surveiller le SUT
- Simuler ou émuler l'environnement du SUT

La couche d'adaptation de test fournit les fonctionnalités suivantes:

- Médiation entre les définitions de test neutres technologiquement et les exigences technologiques spécifiques du SUT et des dispositifs de test
- Application de différents adaptateurs de technologie spécifique pour interagir avec le SUT
- Distribution de l'exécution de test à travers de multiples dispositifs/interfaces de test ou exécution de test locale

3.1.6 Gestion de la configuration d'une TAS

Normalement, une TAS est en cours de développement dans diverses itérations/versions et doit être compatible avec les itérations/versions du SUT. La gestion de la configuration d'une TAS peut inclure:

- Des modèles de test
- Des définitions/spécifications incluant les données de test, les cas de test et les librairies
- Des scripts de test
- Des moteurs d'exécution de test et des outils et composants supplémentaires
- Des adaptateurs de test pour le SUT
- Des simulateurs et émulateurs pour l'environnement du SUT
- Les résultats et les rapports de test

Ces éléments constituent le testware et doivent être à la bonne version pour correspondre à la version du SUT. Dans certaines situations il peut être nécessaire de revenir aux précédentes versions du SUT, par exemple, dans le cas où les problèmes en environnement opérationnel doivent être reproduits avec des versions du SUT plus anciennes. Une bonne gestion de la configuration le rend possible.

3.1.7 Gestion du projet d'une TAS

Comme tout projet d'automatisation de test est un projet logiciel, la même gestion de projet que n'importe quel projet logiciel est requise. Un TAE a besoin d'effectuer les tâches pour toutes les phases du cycle de vie lorsque qu'il développe la TAS. Aussi, un TAE a besoin de comprendre que l'environnement de développement de la TAS devrait être conçu de telle sorte que les informations de statut (métriques) puissent être extraites facilement ou automatiquement rapportées à la gestion du projet de la TAS.

3.1.8 Support de la TAS pour la gestion de test

Une TAS doit apporter un support de gestion du test pour le SUT. Des rapports de test incluant les enregistrements des tests et les résultats de test doivent être facilement extractibles ou automatiquement fournis à la gestion du test (personne ou système) du SUT.

3.2 Conception de la TAA

3.2.1 Introduction à la conception de la TAA

Il y a un nombre d'étapes requit pour concevoir une TAA. Ces étapes sont abordées dans les sections qui suivent. Plus ou moins d'étapes peuvent être nécessaires selon la complexité de la TAA.

Capturer les exigences nécessaires pour définir une TAA appropriée
Les exigences pour une solution d'automatisation de test doivent prendre en compte ce qui suit:

- Quelle activité ou phase du processus de test devrait être automatisée, par exemple, la gestion, la spécification, la génération et l'exécution du test
- Quel niveau devrait être supporté, par exemple, le niveau composant, intégration, système
- Quel type de test devrait être fait, par exemple, test fonctionnel, test de conformité, test d'interopérabilité
- Quel rôles de test devraient exister, par exemple, exécuteur de test, concepteur de test, gestionnaire de qualité de test, gestionnaire de test
- Quel produit logiciel, ligne de produit logiciel, famille de produit logiciel devrait être supporté, par exemple, pour définir la portée et la durée de vie de la TAS implémentée
- Quelles technologies de SUT devraient être supportées, par exemple, pour définir les technologies de SUT avec lesquelles la TAS doit être compatible

Comparer et contraster les différentes approches de conception/architecture
Le TAE doit analyser les pour et les contre des différentes approches lors de la conception des couches sélectionnées de la TAA. Ils incluent mais ne sont pas limités à:

- Considérations pour la couche de génération de test :
  - Sélection de la génération de test manuelle ou automatisée
  - Sélection de génération de test axée sur les exigences, les données, le scenario, ou le comportement
  - Sélection de stratégies de génération de test (par exemple, arbres de classification pour les approches axées sur les données, utilisation de la couverture des cas de test/cas d'exception pour celles basées sur le scénario, transition/état/chemins pour celle basée sur le comportement, etc.)
  - Choix de la stratégie de sélection des tests. En pratique, une génération de tests combinés complète est infaisable comme cela mène à l'explosion du nombre de cas de test. Par conséquent, les critères de couverture pratiques, les poids, les évaluations de risque, etc. devraient être utilisés pour guider la génération de test et la sélection de test ultérieures
- Considérations pour la couche de définition de test:
  - Sélection de définition de test axée sur les données, les mots-clés, les patterns, ou les modèles
  - Sélection de notation pour la définition des tests (par exemple, tables, notations basées sur les états, notation stochastique, notation basée sur le flux de données, sur les processus Métier, axée sur le scénario, etc. par l'utilisation de logiciels comme les tableurs, les langages de test spécifiques à un domaine, TTCN-3, UTP, etc.)
  - Sélection de guides de style et de lignes directrices pour la définition de tests de haute qualité
  - Sélection de répertoires de cas de test (tableurs, bases de données, fichiers, etc.)
- Considérations pour la couche d'exécution de test:
  - Sélection d'outils d'exécution de test
  - Sélection d'une approche interprétation (par l'utilisation d'une machine virtuelle) ou compilation pour implémenter les procédures de test – ce choix dépend généralement de l'outil d'exécution de test choisi
  - Sélection de la technologie d'implémentation pour mettre en œuvre les procédures de test (impérative, telle que C; fonctionnelle, telle que Haskell ou Erlang; axée sur l'objet, telle que C++ , C#, Java; scripting, telle que Python ou Ruby, ou une technologie spécifique à un objet) – ce choix dépend généralement de l'outil d'exécution de test
  - Sélection de librairies d'aide pour faciliter l'exécution de test (par exemple, librairies de dispositif de test, librairie de codage/décodage, etc.)
- Considérations pour la couche d'adaptation de test:

- Sélection d'interfaces de test du SUT
- Sélection d'outils pour simuler et observer les interfaces de test
- Sélection d'outils pour surveiller le SUT pendant l'exécution de test
- Sélection d'outils pour tracer l'exécution de test (par exemple, incluant le timing de l'exécution de test)

Identifier les domaines où l'abstraction peut offrir des avantages
L'abstraction dans une TAA permet l'indépendance technologique de sorte que la même suite de test puisse être utilisée dans différents environnements, et dans différentes technologies cibles. La portabilité des artefacts de test augmente. De plus, la neutralité commerciale est assurée ce qui évite les effets de blocage pour une TAS. L'abstraction améliore également la maintenabilité et l'adaptabilité à des nouvelles technologies de SUT ou des technologies de SUT en évolution. En outre, l'abstraction contribue à rendre la TAA (et ses instanciations par TAS) plus accessibles aux non-techniciens comme les suites de test peuvent être documentées (incluant des moyens graphiques) et expliquées à un niveau plus élevé, ce qui améliore la lisibilité et la compréhension.

Le TAE doit discuter avec les parties prenantes dans le développement du logiciel, dans l'assurance qualité et les tests, quel niveau d'abstraction utiliser dans quels domaines de la TAS. Par exemple, quelles interfaces des couches de l'adaptation de test et/ou de l'exécution de test doivent être externalisées, formellement définies et gardées stable tout au long de la durée de vie de la TAA ? Doit être également discuté si une définition de test abstraite va être utilisée ou si la TAA utilise une couche d'exécution de test avec des scripts de test uniquement. De même, il doit être décidé si la génération de test est abstraite par l'utilisation de modèles de test et d'approches basées sur les modèles. Le TAE doit être conscient qu'il faut faire des compromis entre les implémentations sophistiquées et simples d'une TAA en respectant la capacité fonctionnelle, la maintenabilité et l'extensibilité. Décider du niveau d'abstraction à utiliser dans une TAA nécessite de prendre en compte ces compromis.

Plus l'abstraction est utilisée pour une TAA, plus elle est flexible tout en respectant des évolutions futures ou des transitions vers ces nouvelles approches ou technologies. Cela entraine des investissements initiaux plus élevés (par exemple, architecture et outils d'automatisation de test plus complexes, compétences requises plus grandes, courbes d'apprentissage plus grandes), ce qui retarde le seuil de rentabilité initial mais peut se révéler payant sur du long terme.

Alors que les considérations de ROI (retour sur investissement) détaillées sont à la charge du TAM, le TAE doit fournir des entrées pour analyser le ROI en fournissant des évaluations techniques et des comparaisons des différentes approches et architectures d'automatisation de test sur le timing, les coûts, les efforts et les bénéfices.

Comprendre les technologies des SUT et comment ils s'interconnectent avec la TAS
L'accès aux interfaces de test du SUT est essentiel à toute exécution de test automatisé. L'accès peut se faire aux niveaux suivants :

- Niveau Logiciel, p.ex., le SUT et le logiciel de test sont liés ensemble
- Niveau API, p.ex., la TAS invoque les fonctions/opérations/méthodes fournies par une interface de programmation d'application (à distance)
- Niveau Protocole, p.ex., la TAS interagit avec le SUT via HTTP, TCP, etc.
- Niveau Service, p.ex., la TAS interagit avec les services du SUT via les Web Services, Services RESTful, etc.

De plus, le TAE doit décider quel modèle de TAA doit être utilisé pour interagir entre la TAA et le SUT, même si la TAS et le SUT sont séparés par des API, des protocoles ou des services. Ces paradigmes incluent les éléments suivants:

- Paradigme piloté par les évènements, qui crée l'interaction au travers des évènements échangés sur un bus d'évènements
- Paradigme Client-serveur, qui fait l'interaction par l'invocation de service depuis le fournisseur du service vers le demandeur de service
- Paradigme Pair-à-pair , qui conduit l'interaction via l'invocation de service depuis un pair

Souvent le choix du paradigme dépend de l'architecture du SUT et peut avoir des implications sur l' architecture du SUT. L'interconnexion entre le SUT et la TAA doit être analysée précisément et conçue de façon à sélectionner une architecture future sécurisée entre les deux.

Comprendre l'environnement du SUT

Un SUT peut être un logiciel autonome ou un logiciel qui fonctionne uniquement en lien à d'autres logiciels (par exemple, systèmes de systèmes), matériel (p. ex., systèmes embarqués) ou composants environnementaux (par exemple, système cyber-physique). Une TAS simule ou émule l'environnement SUT dans le cadre de la mise en place d'un test automatisé.

Voici des exemples d'environnements de test et de leur utilisation:

- Un ordinateur avec le SUT et la TAS ensemble – utile pour tester une application logicielle
- Ordinateurs individuels SUT et TAS en réseau – utile pour des tests de logiciels de serveur
- Dispositifs de test supplémentaires pour stimuler et observer les interfaces techniques d'un SUT – utile pour tester le logiciel d'un boitier décideur (set top box)
- Des dispositifs en réseau pour émuler l'environnement opérationnel du SUT – utile pour tester le logiciel d'un routeur réseau
- Des simulateurs pour simuler l'environnement physique du SUT– utile pour tester le logiciel d'un composant de contrôle intégré

Temps et complexité pour l'implémentation d'une architecture de test
Alors que l'estimation de l'effort pour un projet de TAS est de la responsabilité d'un TAM, le TAE doit support un TAM en fournissant de bonnes estimations sur le temps et la complexité de la conception d'une TAA. Voici des méthodes d'estimation et des exemples:

- Estimation basée sur l'analogie comme l'analyse des points de fonction, estimations à 3 points, Wide Band delphi, estimation basée sur l'expertise
- Estimation en utilisant les structures de décomposition du travail comme celles des logiciels de gestion ou des modèles projet
- Estimation avec des paramètres utilisant "Constructive Cost Model (COCOMO)"
- Estimations basées sur la taille utilisant l'analyse des points de fonction, l'analyse des points de User Story ou l'analyse des cas d'utilisation
- Les estimations de groupe utilisant le Planning Poker

Facilité d'utilisation d'une implémentation d'architecture de test
En plus des fonctionnalités de la TAS, de sa compatibilité avec le SUT, de sa stabilité à long terme et son évolutivité, des efforts requis et des considérations de ROI, le TAE a la responsabilité spécifique de traiter les problèmes d'utilisabilité de la TAS. Cela inclut, mais n'est pas limité à:

- Conception orientée testeur

- Facilité d'utilisation de la TAS
- Support de la TAS pour d'autres acteurs du développement logiciel, assurance qualité et gestion de projet
- Organisation efficace, navigation, et recherche dans/avec la TAS
- Documentation utile, manuels et texte d'aide pour la TAS
- Reporting pratique par et au sujet de la TAS
- Conception itérative pour traiter les feedback et avancer empiriquement sur la TAS

3.2.2 Approches pour automatiser les cas de test

Les cas de test doivent être traduits en séquences d'actions qui sont exécutées sur le SUT. Cette séquence d'actions peut être documentée dans une procédure de test et/ou peut être implémentée dans un script de test. En plus des actions, les cas de test automatisés devraient également définir les données de test pour l'interaction avec le SUT et inclure les étapes de vérification pour vérifier que le résultat attendu est atteint par le SUT. Plusieurs approches peuvent être utilisées pour créer une séquence d'actions:

1. Le TAE implémente les cas de test directement en scripts de test automatisés. Cette option est la dernière recommandée car elle manque d'abstraction.
2. Le TAE conçoit les procédures de test et les transforme en scripts de test automatisés. Cette option a de l'abstraction mais manque d'automatisation pour générer les scripts de test.
3. Un outil traduit les procédures de test en des scripts de test automatisés. Cette option combine à la fois abstraction et génération de script automatisé.
4. Un outil génère les procédures de test automatisées et/ou les données de test depuis les modèles. Cette option a le plus haut degré d'automatisation.

Notez que les options disponibles dépendent fortement du contexte du projet. Il peut également être efficace de commencer l'automatisation de test en appliquant une des options les moins avancées, car elles sont faciles à mettre en œuvre, et peuvent avoir une valeur ajoutée à court terme bien que cela résultera dans une solution moins maintenable.

Des approches bien établies pour l'automatisation des cas de test incluent:

- Approche capture/rejeu, qui peut être utilisée pour l'option 1
- Approche de scripting structurée, approche pilotée par les données et approche pilotée par les mots-clés, qui peuvent être utilisées pour les options 2 ou 3
- Test basé sur les modèles (incluant l'approche pilotée par les processus Métier), qui peut être utilisé pour l'option 4

Ces approches sont expliquées par la suite en termes de concepts principaux avec les avantages et les inconvénients.

Approche capture/rejeu

Concept Principal

Dans l'approche capture/rejeu, les outils sont utilisés pour capturer les interactions avec le SUT lorsqu'une séquence d'actions définies par une procédure de test est réalisée. Les entrées sont capturées; les sorties peuvent également être enregistrées pour des vérifications futures. Pendent le rejeu des évènements, il existe différentes possibilités de vérification, manuelles ou automatisées:

- Manuelle: Le testeur doit regarder les sorties du SUT pour détecter les anomalies

- Complète: Toutes les sorties du système qui ont été enregistrées pendant la capture doivent être reproduites par le SUT
- Exacte: Toutes les sorties du système qui ont été enregistrées pendant la capture doivent être reproduites par le SUT au niveau de détail enregistré
- Checkpoints: Seulement les sorties du système sélectionnées sont vérifiées à un certain moment avec des valeurs spécifiées

Avantages

L'approche capture/rejeu peut être utilisée pour les SUTs au niveau IHM et/ou au niveau API. Dans un premier temps, c'est facile à mettre en place et à utiliser.

Inconvénients

Les scripts de capture/rejeu sont difficiles à maintenir et à faire évoluer car l'exécution capturée du SUT dépend fortement de la version de SUT. Par exemple, en enregistrant au niveau IHM, les changements d'IHM peuvent impacter le script de test, même s'il s'agit seulement du positionnement d'un élément graphique. Les cas de test basés sur la capture/rejeu sont très fragiles.

L'implémentation des cas de test (scripts) peut seulement commencer quand le SUT est disponible.

Scripting linéaire

Concept Principal

Comme avec toutes les techniques de scripting, le scripting linéaire commence avec des procédures de test manuelles. Notez cependant que cela peut ne pas être des documents écrits – Savoir quels tests exécuter et comment les exécuter peut être connu par un ou plusieurs Analystes de Test.

Chaque test est exécuté manuellement pendant que l'outil de test enregistre la séquence d'actions et dans certains cas capture les sorties visibles à l'écran du SUT. Cela résulte en général dans un script (typiquement long) pour chaque procédure de test. Les scripts enregistrés peuvent être édités pour améliorer la lisibilité (c.à.d., en ajoutant des commentaires pour expliquer ce qui se passe aux points clés) ou ajouter d'autres vérifications en utilisant le langage de scripting de l'outil.

Les scripts peuvent alors être rejoués par l'outil, faisant répéter à l'outil les mêmes actions que celles faites par le testeur quand le script a été enregistré. Bien que cela puisse être utilisé pour automatiser les tests d'IHM, ce n'est pas une bonne technique à utiliser quand un grand nombre de tests sont à automatiser et seront requis pour plusieurs versions du logiciel. C'est à cause des couts de maintenance élevés généralement dus aux changements du logiciel sous test (chaque changement dans le SUT peut entrainer beaucoup de changements dans les scripts enregistrés.

Avantages

Les avantages des scripts linéaires se focalisent sur le fait qu'il y a peu ou pas de travail de préparation avant de pouvoir automatiser. Une fois que vous avez appris à utiliser l'outil, il n'y a qu'à enregistrer un test manuel et à le rejouer (bien que la partie enregistrement puisse nécessiter une interaction supplémentaire avec l'outil de test pour demander des comparaisons entre les résultats obtenus avec les résultats attendus et vérifier que le logiciel fonctionne correctement). Des compétences en programmation ne sont pas exigées, mais sont généralement utiles.

Inconvénients

Les désavantages des scripts linéaires sont nombreux. La somme d'effort demandée pour automatiser une procédure de test donnée sera fortement dépendante de sa taille (nombre d'étapes ou d'actions). Donc, la 1000ème procédure de test à automatiser prendra un temps proportionnellement similaire au temps passé à automatiser la 100ème procédure. En d'autres termes, il n'y a pas beaucoup de possibilités de diminuer le cout de fabrication des nouveaux tests automatisés.

Par ailleurs, s'il y avait un second script qui exécute un test similaire avec différentes valeurs, ce script contiendrait la même séquence d'instructions que le premier script; seule l'information incluse avec les instructions (c'est à dire les arguments d'instruction ou les paramètres) serait différente. S'il y avait plusieurs tests (et donc les scripts) ils contiendraient tous les mêmes instructions, qui toutes nécessiteraient d'être maintenues lorsque le logiciel a un changement qui affecte les scripts.

Comme les scripts sont dans un langage de programmation, plutôt que dans un langage naturel, les non-programmeurs peuvent les trouver difficiles à comprendre. Certains outils de test utilisent des langages propriétaires (unique à l'outil) donc cela prend du temps d'apprendre le langage et de devenir expérimenté dans celui-ci.

Les scripts enregistrés ne contiennent que des instructions générales dans les commentaires, quand il y en a. Les longs scripts en particulier sont mieux annotés avec des commentaires pour expliquer ce qui se passe à chaque étape du test. Cela rend la maintenance plus facile. Les scripts peuvent vite devenir de taille importante (contenant de nombreuses instructions) quand le test est composé de nombreuses étapes.

Les scripts ne sont pas modulaires et difficiles à maintenir. Les scripts linéaires ne respectent pas les paradigmes communs de réutilisabilité et de modularité et sont liés étroitement à l'outil utilisé.

Scripting structuré

Concept Principal

La différence majeure entre la technique de scripting structuré et celle de scripting linéaire est l'introduction d'une bibliothèque de scripts. Elle contient des scripts réutilisables qui exécutent des séquences d'instructions qui sont habituellement requises par de nombreux tests. Un bon exemple de ces scripts sont ceux qui interfacent avec le SUT.

Avantages

Les bénéfices de cette approche incluent une réduction significative de la maintenance et des couts réduits d'automatisation de nouveaux tests (car ils peuvent utiliser des scripts qui existent déjà plutôt que d'avoir à les créer de rien).

Les avantages des scripts structurés sont largement atteints par la réutilisabilité des scripts. Plus de tests peuvent être automatisés sans avoir à créer le volume de scripts qu'une approche de scripting linéaire demanderait. Cela a un impact direct sur les couts de maintenance et de fabrication. Le second test et les suivants ne prendront pas autant d'effort à automatiser car le script créé pour implémenter le premier test peut être réutilisé à nouveau.

Inconvénients

L'effort initial pour créer des scripts partagés peut être vu comme un désavantage mais cet investissement initial devrait renvoyer des bénéfices si l'approche est menée correctement. Des

compétences en programmation sont requises pour créer tous les scripts puisque simplement enregistrer n'est pas suffisant. La bibliothèque de scripts doit être bien gérée, c.à.d.. les scripts devraient être documentés et l'analyste Technique de Test devrait trouver les scripts requis facilement (donc une bonne convention de nommage serait pertinente ici).

Tests pilotés par les données

Concept Principal

La technique de scripting pilotée par les données est basée sur la technique de scripting structure. La plus grande différence est comment les données d'entrée sont prises en charge. Les entrées sont extraites des scripts et places dans un ou plusieurs fichiers séparés (appelés fichiers de données).

Cela signifie que le script principal peut être réutilisé pour implémenter plusieurs tests (plutôt qu'un simple test). Typiquement, le script principal réutilisable est appelé script de contrôle. Le script de control contient la séquence d'instructions nécessaires pour exécuter les tests mais les données d'entrée dans le fichier de données. Un test de contrôle peut être utilisé pour plusieurs tests mais cela est habituellement insuffisant pour automatiser une grande quantité de tests. Donc un nombre de scripts de contrôle sera requis mais ils ne seront qu'une fraction des tests qui seront automatisés.

Avantages

Le cout de l'ajout de nouveaux tests peut être réduit de façon significative par cette technique de scripting. Cette technique est utilisée pour automatiser plusieurs variations de tests importants, apportant une plus grande profondeur de test plutôt qu'une plus grande couverture.

Avoir des tests 'décrits' par les fichiers de données signifie que l'Analyste de Test peut spécifier des tests automatisés simplement par un ou plusieurs fichiers de données. Cela donne à l'Analyste de Test plus de liberté pour spécifier les tests automatisés sans être très dépendant de l'Analyste Technique de Test (qui peut être une ressource rare).

Inconvénients

Un désavantage est la gestion des fichiers de données qui doivent être lus correctement par la TAS, mais cela peut se gérer de manière appropriée.

Des cas de test négatifs importants peuvent être oubliés également. Les tests négatifs sont une combinaison de procédures de test et de données de test. Dans une approche ciblant principalement les données de test, des «procédures de test négatives» peuvent être manquées.

Test piloté par les mots-clés

Concept Principal

La technique de scripting piloté par les mots-clés est basée sur la technique de scripting pilotée par les données. Il y a deux différences principales: (1) Les fichiers de données sont appelés fichiers de 'définition de test' ou un nom similaire (p.ex., Fichier des mots d'Action); et (2) il n'y a qu'un seul script de contrôle.

Un fichier de définition de test contient la description des tests d'une façon qui permet à l'Analyste de Test de comprendre facilement (plus facilement qu'un fichier de données équivalent). Il contient habituellement les données comme le ferait un fichier de données, mais le fichier de mots-clés

contient également les instructions de haut niveau (les mots-clés, ou 'mots d'action').

Les mots-clés devraient être choisis de façon à être pleinement compréhensibles par l'analyste de test, les tests étant décrits et l'application étant testée. Ils sont le plus souvent (mais pas exclusivement) utilisés pour représenter des interactions métier de haut niveau avec le système (par exemple, passer une commande). Chaque mot-clé représente un nombre d'interactions détaillées avec le SUT. Les séquences de mots-clés (incluant les données) sont utilisées pour spécifier les cas de test. Des mots-clés spéciaux peuvent être utilisés pour des étapes de vérification, ou les mots-clés peuvent contenir à la fois les mots-clés et les étapes de vérification.

La responsabilité de l'analyste de test comprend la création et la maintenance des fichiers de mots-clés. Cela signifie qu'une fois les scripts implémentés, les analyses de tests peuvent ajouter des tests 'automatisés' simplement en les spécifiant dans un fichier de mots-clés (comme dans un script piloté par les données).

Avantages

Une fois que les scripts de contrôle et que les scripts pour les mots-clés ont été écrits, le cout de l'ajout de nouveaux tests automatisés sera beaucoup réduit par cette technique de scripting.

Avoir les tests 'décrits' par les fichiers de mots-clés signifie que les analystes de test peuvent spécifier des tests 'automatisés' simplement en décrivant les tests par les mots-clés et les données associées. Cela donne aux analystes de tests plus de libertés pour spécifier des tests automatisés sans être très dépendants de l'analyste technique de test (qui peut être une ressource rare). Le bénéfice de l'approche pilotée par les mots-clés par rapport à l'approche pilotée par les données est ici l'utilisation des mots-clés. Chaque mot-clé devrait représenter une séquence d'actions détaillées qui produisent des résultats compréhensibles. Par exemple, 'créer un compte', 'passer une commande', 'vérifier le statut de la commande', sont toutes des actions possibles pour une application d'achat en ligne qui chacune implique un nombre d'étapes détaillées. Quand un analyste de test décrit un test système à un autre analyste de test, il y a de grandes chances qu'il parle avec les termes d'actions de haut niveau et non avec les étapes détaillées. Le but de l'approche pilotée par les mots-clés est alors d'implémenter ces actions de haut niveau et de permettre aux tests d'être définis en terme d'actions de haut niveau sans référence aux étapes détaillées.

Ces cas de test sont plus faciles à maintenir, à lire et à écrire car la complexité peut être cachée dans le mot-clé (ou dans les bibliothèques, dans le cas d'une approche de scripting structurée). Les mots-clés peuvent offrir une abstraction de la complexité des interfaces du SUT.

Inconvénients

Implémenter les mots-clés reste une tâche importante d'automatisation par les ingénieurs de test, particulièrement si l'outil utilisé n'offre pas de support pour cette technique de scripting. Pour les petits systèmes cela peut entrainer des frais importants d'implémentation et le cout sera supérieur au bénéfice.

Il faut faire attention de s'assurer que les mots-clés corrects sont implémentés. De bons mots-clés seront souvent utilisés dans différents tests alors que des mots-clés pauvres ont de grandes chances d'être peu utilisés.

Scipting piloté par les processus

Concept principal

L'approche pilotée par les processus est construite sur la politique de script pilotée par les mots-clés à la différence que les scénarios - représentant les cas d'utilisation du logiciel en test - constituent des scripts paramétrés avec les données ou combinés en définitions de test de haut niveau.

Ces définitions de tests sont plus faciles à traiter, comme la relation logique entre les actions. Par exemple, “vérifier l'état de la commande” après “passer la commande” dans les tests de fonctions ou “vérifier l'état de la commande” sans au préalable “passer la commander" dans les tests de robustesse.

Avantages

L'utilisation de définition de cas de test basée sur un scénario semblable au processus permet aux procédures de test d’être définies comme un workflow. L'objectif de l'approche pilotée par les processus est de mettre en œuvre ces workflow de haut niveau en utilisant des bibliothèques de tests qui contiennent les étapes de test détaillées (voir aussi l'approche pilotée par les mots-clés).

Inconvénients

Les processus d’un SUT peuvent ne pas être faciles à comprendre par un Analyste Technique de Test – et donc l’implémentation de ces scripts orientés processus, en particulier si aucun processus Métier logique n’est supporté pas l’outil.

Il faut également s’assurer que les processus corrects, par l’utilisation de mots-clés corrects sont mis en œuvre. De bons processus seront référencés par d’autres ce qui résultera en de nombreuses zones testées alors que des processus de mauvaise qualité n’apporteront pas grand-chose en terme de pertinence, capacité à détecter des erreurs, etc.

Test basé sur les modèles (MBT)

Concept Principal

Le test basé sur les modèles se réfère à la génération automatique des cas de test (voir aussi le syllabus Model-Based Testing de l’ISTQB®) – par opposition à l’exécution automatisée de cas de test – en utilisant la capture/rejeu, le scripting linéaire, le scripting structuré, le scripting piloté par les données ou le scripting piloté par les processus.
Le MBT utilise des modèles (semi-) formels avec abstraction de la technologie de scripting de la TAA. Différentes méthodes de génération de test peuvent être utilisées pour dériver les tests quel que soit le Framework de scripting discuté précédemment.

Avantages

Le MBT permet grâce à son abstraction de se concentrer sur l’essence du test (en termes de logique métier, données, scenarios, configurations, etc.). Il permet également la génération de tests pour différents systèmes et technologies cibles, de telle façon que les modèles utilisés pour la génération de test constituent un testware sécurisé qui pourra être réutilisé et maintenu lorsque la technologie évoluera.

En cas de changements dans les exigences, le modèle de test doit être seulement adapté; un ensemble complet de cas de test est généré automatiquement. Ces techniques de conception de test sont intégrés dans les générateurs de cas de test.

Inconvénients

Une expertise en modélisation est requise pour réaliser une approche MBT efficacement. La tâche de modélisation par abstraction des interfaces, des données et/ou du comportement du SUT peut être difficile. De plus, les outils de modélisation et de MBT ne sont pas pour le moment très présents, mais gagnent en maturité. L'approche MBT demande un ajustement des processus de test. Par exemple, le rôle du concepteur de test doit être établi. De plus, les modèles utilisés pour la génération des tests sont des éléments majeurs de l'assurance qualité d'un SUT et leur qualité et leur maintenance doivent être vérifiées.

3.2.3 Considérations techniques d'un SUT

Les aspects d'un SUT doivent être pris en compte pour concevoir la TAA. Certains sont discutés ci-dessous bien que ce ne soit pas une liste complète mais des exemples d'aspects importants.

Interfaces du SUT
Un SUT possède des interfaces internes (dans le système) et des interfaces externes (vers l'environnement du système et ses utilisateurs ou par des composants exposés). Une TAA doit être capable de contrôler et/ou d'observer toutes ces interfaces du SUT qui sont potentiellement affectées par les procédures de test (c.à.d. les interfaces doivent être testables). De plus, il peut être nécessaire d'enregistrer les interactions entre le SUT et la TAS à différents niveaux de détails, comme typiquement les horodatages.

Se focaliser sur les tests (p.ex., un test) est nécessaire en début de projet (ou continuellement dans les environnements Agile) pendant la définition de l'architecture pour vérifier la disponibilité des interfaces de test nécessaires ou des aménagements à apporter au SUT pour le rendre testable (Conception pour testabilité).

Données du SUT
Un SUT utilise des données de configuration pour contrôler l'instanciation, la configuration, l'administration, etc. De plus, il utilise les données utilisateur qu'il traite. Un SUT peut également utiliser des données externes d'autres systèmes pour réaliser ses tâches. En fonction des procédures de test d'un SUT, tous ces types de données doivent être définissables, configurables et instanciables par la TAA. La façon de traiter les données du SUT est décidée pendant la conception de la TAA. En fonction de l'approche, les données peuvent êtres des paramètres, des feuilles de données de test, des bases de données de test, des données réelles, etc.

Configurations du SUT
Un SUT peut être déployé dans différentes configurations, par exemple sur différents systèmes d'exploitation, sur différents dispositifs cibles, ou avec différents langages. En fonction des procédures de test, différentes configurations de SUT peuvent être traitées par la TAA. Les procédures de test peuvent nécessiter différentes mises en place des tests (dans un labo) ou des mises en place virtuelles (dans le cloud) de la TAA en combinaison avec une configuration de SUT donnée. Il peut être nécessaire d'ajouter des simulateurs et/ou des émulateurs de composants du SUT pour certains aspects.

Normes et éléments juridiques
En plus des aspects techniques d'un SUT, la conception de la TAA peut devoir respecter des exigences normatives ou légales. Les exigences de confidentialité des données de test et de secret peuvent par exemple avoir des impacts sur les fonctions d'enregistrement et de reporting de la TAA.

Outils et environnements outillés utilisés pour développer le SUT
Au long du développement d'un SUT, différents outils peuvent être utilisés pour l'ingénierie des exigences, la conception et la modélisation, le codage, l'intégration et le déploiement du SUT. La TAA avec ses propres

outils devrait prendre en compte l'ensemble des outils du SUT de façon à permettre la compatibilité, traçabilité et/ou la réutilisation d'artefacts.

Interfaces de test dans le produit logiciel
Il est fortement recommandé de ne pas supprimer toutes les interfaces de test avant la release du produit. Dans la plupart des cas, ces interfaces peuvent être laissées dans le SUT sans causer de dommage au produit final. Si elles restent en place, les interfaces peuvent être utilisées par les ingénieurs des services de support pour le diagnostic comme pour le test des nouvelles versions en maintenance. Il est important de vérifier que les interfaces ne créeront pas de risques de sécurité. Si nécessaire, les développeurs peuvent habituellement désactiver les interfaces de test de façon à ce qu'elles ne puissent pas être utilisées en dehors du département de développement.

3.2.4 Considérations pour les processus de développement/Assurance Qualité

Les aspects de développement et d'assurance qualité d'un SUT doivent être pris en compte pendant la conception de la TAA. Certains de ces aspects sont discutés ci-dessous dans une liste qui n'est pas exhaustive mais présente des exemples d'aspects importants.

Exigences de contrôle d'exécution de test
En fonction du niveau d'automatisation requis par la TAA, l'exécution de tests interactive, l'exécution de tests en mode batch ou l'exécution de tests entièrement automatisés peuvent devoir être pris en charge par la TAA.

Exigences de Reporting
En fonction des exigences de reporting incluant les types de rapports et leurs structures, la TAA doit pouvoir prendre en compte des rapports de test fixes, paramétrables, ou définis dans plusieurs formats et mises en forme.

Rôles et droit d'accès
En fonction des exigences de sécurité, la TAA peut devoir prendre en charge plusieurs rôles et droits d'accès.

Panorama des outils en place
Le panorama des outils mis en place peut comprendre les outils de gestion de projet, de gestion de test, les référentiels de code et de test, la gestion des anomalies, des incidents, l'analyse des risques, etc... La TAA est également outillée par un ou plusieurs outils qui doivent s'intégrer avec les autres outils du panorama. De plus, les scripts de test devraient être stockés et versionnés comme le code du SUT pour que les révisions suivent le même processus pour les deux.

3.3 Développement de la TAS

3.3.1 Introduction au développement de la TAS

Le développement de la TAS est comparable à d'autres projets de développement d'autres logiciels. Il peut suivre les mêmes procédures et processus incluant les revues par les pairs par les développeurs et les testeurs. La synchronisation et la compatibilité avec la TAS sont spécifiques à une TAS. La conception de la TAA doit le prendre en compte (voir Section 3.2) ainsi que le développement de la TAS.

Cette section utilise le cycle de vie de développement logiciel (SDLC) pour expliquer le processus de développement de la TAS et les aspects processus de compatibilité et de synchronisation au SUT. Ces aspects sont également importants pour tout processus de développement qui a été choisi ou est en place pour le développement du SUT et/ou de la TAS – Ils doivent être adaptés en conséquence.

Le SDLC basique pour une TAS est montré Figure 2.

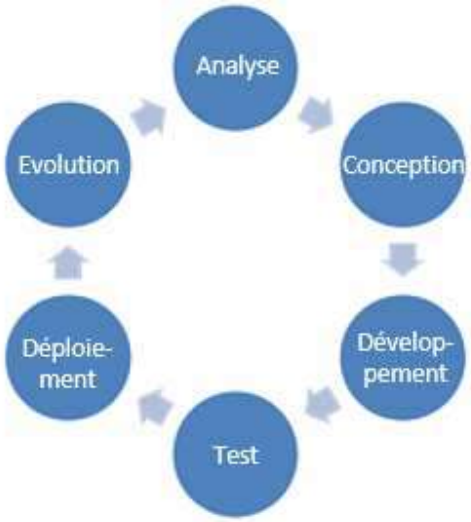

Figure 2: SDLC basique pour une TAS

L'ensemble des exigences d'une TAS doit être analysé et collecté (voir Figure 2). Les exigences guident la conception de la TAS comme définie par son architecture (voir Section 3.2). La conception est transformée en logiciel par l'ingénierie logicielle. Merci de noter qu'une TAS peut également utiliser des dispositifs de test matériels, ce qui est hors périmètre de ce syllabus. Comme tout autre logiciel, la TAS doit être testée. C'est typiquement fait par des tests de capacité basiques de la TAS, suivis par des interactions entre la TAS et le SUT. Après déploiement et utilisation de la TAS, des évolutions sont souvent nécessaires pour ajouter plus de capacités de test, changer les tests ou pour mettre à jour la TAS suite à des évolutions du SUT. L'évolution de la TAS requiert une nouvelle phase de développement en accord avec le SDLC.

Merci de noter que le SDLC ne montre pas la sauvegarde, l'archivage et la suppression de la TAS. Comme pour le développement de la TAS, ces procédures devraient suivre les méthodes établies.

3.3.2 Compatibilité entre la TAS et le SUT

Compatibilité des processus
Le test d'un SUT doit être synchronisé avec son développement – et, dans le cas de l'automatisation des tests, synchronisé avec le développement de la TAS. Par conséquent, il est avantageux de coordonner les processus de développement du SUT, de la TAS et du test. Un gain important est apporté quand le développement du SUT et de la TAS sont compatibles en termes de structure de processus, de gestion de processus et de support d'outil.

Compatibilité des équipes

La compatibilité des équipes est un autre aspect de la compatibilité entre le développement de la TAS et du SUT. Si la façon de manager et l'approche de la TAS et du SUT sont semblables les deux équipes obtiendront des bénéfices en faisant la revue des exigences de chacun, en concevant et/ou développant les artefacts, en discutant des problèmes, et en trouvant des solutions compatibles. La compatibilité des équipes aide également à la communication et aux interactions entre elles.

Compatibilité Technologique
En outre, la compatibilité technologique entre la TAS et le SUT doit être prise en compte. Il est bénéfique de concevoir et de mettre en place une interaction transparente entre la TAS et le SUT dès le départ. Même si cela n'est pas possible (par ex., parce que les solutions techniques ne sont pas disponibles soit pour la TAS soit pour le SUT), interagir de façon transparente par le biais d'adaptateurs, de surcouche ou d'autres formes d'intermédiaires doit être possible.

Compatibilité des outils
La compatibilité des outils pour la gestion, le développement et l'assurance Qualité de la TAS et du SUT doit être prise en compte. Par exemple, si le même outil pour la gestion des exigences et/ou la gestion des problèmes est utilisé, l'échange d'informations et la coordination du développement de la TAS et du SUT sera simplifiée.

3.3.3 Synchronisation entre TAS et SUT

Synchronisation des exigences
Après l'élucidation des exigences, les exigences du SUT et de la TAS doivent être développées. Les exigences de la TAS peuvent être groupées en deux groupes principaux d'exigences: (1) Exigences qui traitent du développement de la TAS comme un système logiciel, comme les exigences fonctionnelles de conception de test, de spécification de test, d'analyse de résultats des tests, etc. et (2) les exigences de test du SUT par la TAS. Ces exigences de test correspondent aux exigences du SUT et reflètent toutes les caractéristiques et les propriétés du SUT qui doivent être testées par la TAS. Chaque fois que les exigences du SUT ou de la TAS sont mises à jour, il est important de vérifier la cohérence entre les deux pour vérifier que toutes les exigences du SUT qui doivent être testées par la TAS ont des exigences de test.

Synchronisation des phases de développement
De façon à avoir la TAS prête quand nécessaire pour tester le SUT, les phases de développement doivent être coordonnées. L'efficacité est meilleure quand les exigences, la conception, la spécification, et les implémentations de la TAS et du SUT sont synchronisées.

Synchronisation de la gestion des défauts
Les défauts peuvent être dus au SUT, à la TAS ou aux exigences/à la conception/aux spécifications. À cause des relations entre les deux projets, si un défaut est corrigé dans l'un, l'action corrective peut impacter l'autre. La gestion des défauts et les re-tests (confirmation) doivent traiter à la fois le SUT et la TAS.

Synchronisation des évolutions du SUT et de la TAS
Le SUT et la TAS peuvent évoluer en même temps pour adapter les nouvelles fonctionnalités ou en supprimer, pour corriger des défauts, ou pour traiter des changements dans leurs environnements (incluant les changements respectifs de la TAS et du SUT car l'un est un composant de l'environnement de l'autre). Tout changement appliqué à un SUT ou une TAS peut impacter l'autre donc la gestion de ces changements devrait concerner à la fois le SUT et la TAS.

Deux approches de synchronisation des processus de développement du SUT et de la TAS sont représentées dans la Figure 3 et la Figure 4.

La Figure 3 montre une approche dans laquelle les deux cycles de vie du SUT et de la TAS sont principalement synchronisés en deux phases: (1) L'analyse de la TAS est basée sur la conception du SUT, qui est lui-même basé sur l'analyse du SUT et (2) le test du SUT qui utilise les TAS déployées.

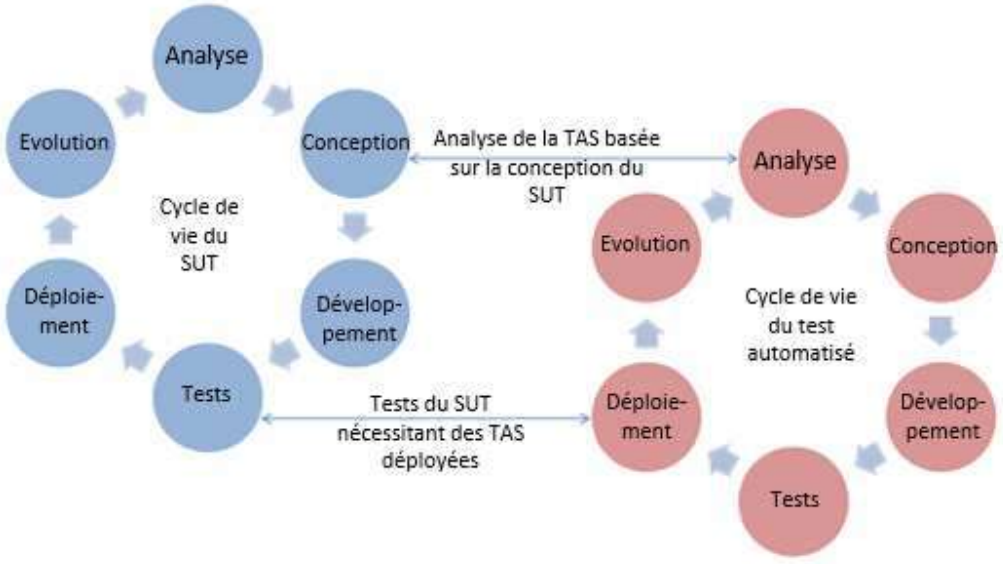

Figure 3: Exemple de synchronisation 1 du processus de développement de la TAS et du SUT

La figure 4 montre une approche hybride avec des tests à la fois manuels et automatisés. Chaque fois que des tests manuels sont faits avant que les tests soient automatisés ou chaque fois que des tests manuels et automatisés sont utilisés ensemble, l'analyse de la TAS doit être basée à la fois sur la conception du SUT et sur les tests manuels. De cette façon, la TAS est synchronisée avec les deux. Le second point majeur de synchronisation pour une telle approche est le même que précédemment: le test du SUT nécessite des tests déployés, ce qui dans le cas de tests manuels, peut juste correspondre aux procédures de test à suivre.

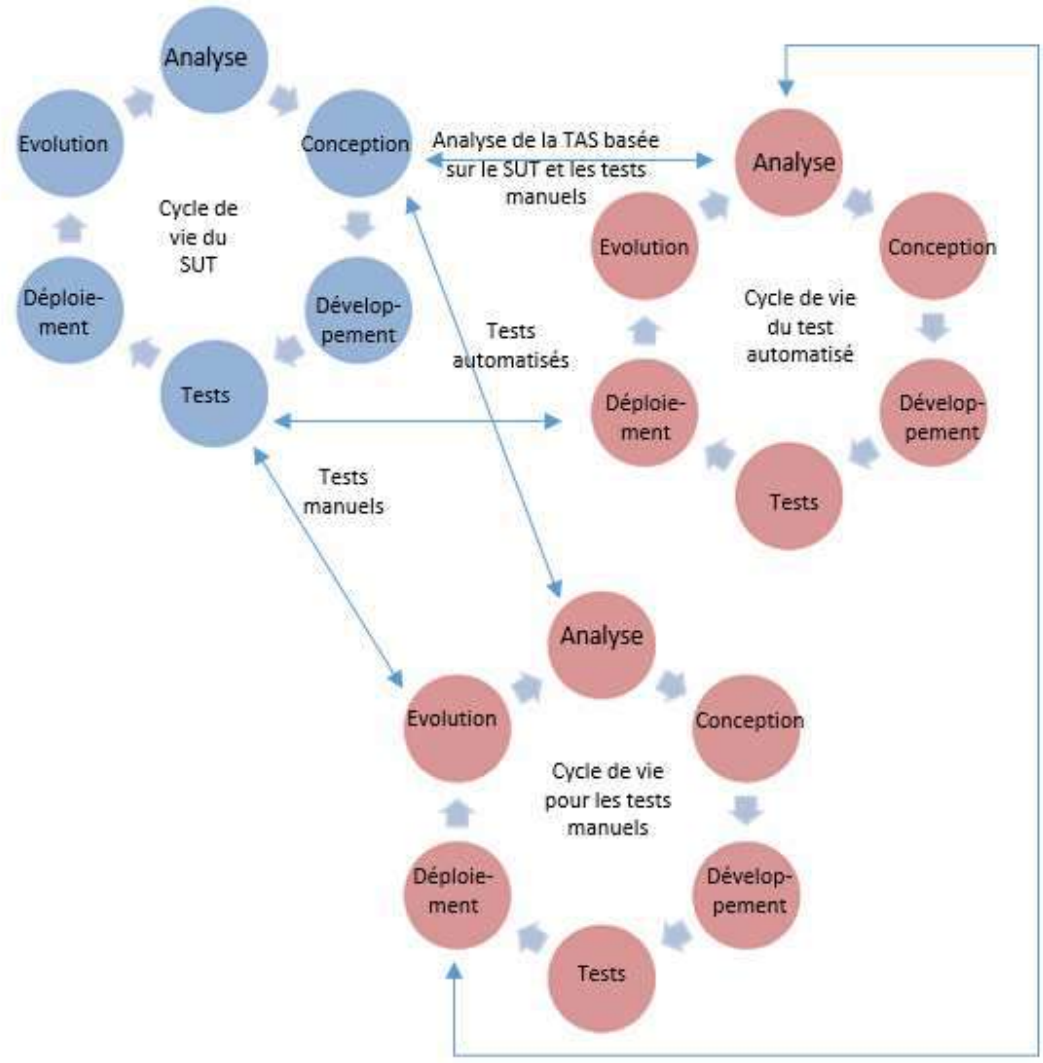

Figure 4: Exemple 2 de synchronisation des processus de développement de la TAS et du SUT

3.3.4 Mettre en œuvre la réutilisation dans la TAS

La réutilisation de la TAS se réfère à la réutilisation des artefacts de la TAS. (Pour tous ses niveaux d'architecture) pour les lignes de produit, les Framework de produits, domaines de produit et/ou familles de projets. Les exigences de réutilisation résultent de la pertinence des artefacts de la TAS pour les autres variantes de produits, produits et/ou projets. Les artefacts de la TAS réutilisables peuvent inclurent:

- (Parties de) modèles d'objectifs de test, scenarios de test, composants de test ou données de test
- (Parties de) cas de test, données de test, procédures de test ou bibliothèques de tests elles-mêmes

- Le moteur de test et/ou le cadre de reporting de test
- Les adaptateurs aux composants du SUT et/ou ses interfaces

Alors que des aspects liés à la réutilisation sont déjà établis quand la TAA est définie, la TAS peut aider à accroitre la capacité de réutilisation en:

- Suivant la TAA ou en la revoyant et en la mettant à jour quand nécessaire
- Documentant les artefacts de la TAS pour qu'ils soient facilement compréhensibles et puissent être incorporés dans de nouveaux contextes
- S'assurant de l'exactitude de tout composant de la TAS pour que leur utilisation dans de nouveaux contextes soit de grande qualité

Il est important de noter que bien que le concept de réutilisation concerne principalement la TAA, la maintenance et les améliorations de cette réutilisabilité se font sur tout le cycle de vie de la TAS. Cela demande un intérêt et un effort continus pour que la réutilisation se fasse, pour mesurer et démontrer la valeur ajoutée de la réutilisation, et pour évangéliser les autres à réutiliser les TAS existantes.

3.3.5 Support à une variété de systèmes cibles

Le support d'une TAS à différents systèmes cibles réfère à la capacité de la TAS à tester différentes configurations de logiciels. Ces différentes configurations concernent les éléments suivants:

- Nombre et interconnexion des composants du SUT
- Environnements (à la fois logiciels et matériels) sur lesquels les composants du SUT tournent
- Les technologies, langages de programmation et systèmes d'exploitation utilisés pour implémenter les composants du SUT
- Les bibliothèques et les packages que les composants du SUT utilisent
- Les outils utilisés pour mettre en œuvre les composants du SUT

Alors que les 4 premiers aspects impactent la TAS à tous niveaux de test, le dernier est le plus pertinent au niveau des tests de composant et d'intégration.

La capacité de la TAS à tester différentes configurations de produits logiciels est déterminée quand la TAA est définie . Cependant, la TAS doit être capable de prendre en charge les différentes variations et capable de gérer les fonctionnalités et les composants nécessaires à ces différentes configurations.

La prise en charge des différentes TAS en relation avec les différents produits logiciels peut être faite avec:

- La gestion de version/configuration de la TAS et du SUT peut être utilisée pour fournir les versions respectives de TAS et de SUT qui se correspondent
- Le paramétrage de la TAS peut être utilisé pour adapter une TAS à une configuration de SUT

Il est important de noter que si la conception de la variabilité de la TAS concerne la TAA, sa maintenance et ses améliorations se font sur tout le cycle de vie de la TAS . Cela requiert un intérêt et des efforts continus pour revoir, ajouter et même supprimer des options et des formes de variabilité.

4 Risques et contingences liés au déploiement - 150 mins.

Mots clés
Atténuation des risques, Risques, Évaluation des risques

Objectifs d'apprentissage pour risques et les contingences liés au déploiement

4.1 Sélection de l'approche d'automatisation des tests et planification du déploiement

ALTA-E-4.1.1 (K3) Appliquer les lignes directrices qui permettent un pilote de test et des activités de déploiement efficaces

4.2 Évaluation des risques et stratégies d'atténuation (mitigation)

ALTA-E-4.2.1 (K4) Analyser les risques de déploiement, identifier les problèmes techniques pouvant mener à l'échec du projet d'automatisation des tests et planifier les stratégies d'atténuation.

4.3 Maintenance des tests automatisés

ALTA-E-43.1 (K2) Comprendre quels facteurs supportent et affectent la maintenabilité de la TAS

4.1 Sélection de l'approche d'automatisation de test et de la planification du déploiement

Deux activités principales sont impliquées dans la mise en œuvre et le déploiement d'une TAS: le pilote et le déploiement. Les étapes qui composent ces deux activités varient en fonction du type de la TAS et de la situation spécifique.

Pour le pilote, les étapes suivantes devraient au moins être envisagées :
- Identifier un projet adapté
- Planifier le pilote
- Conduire le pilote
- Évaluer le pilote

Pour le déploiement, les étapes suivantes devraient au moins être envisagées :
- Identifier le(s) projet(s) cible(s) initial
- Déployer la TAS dans les projets sélectionnés
- Suivre et évaluer la TAS dans les projets après une période prédéfinie
- Déployer sur le reste de l'organisation / projets

4.1.1 Projet pilote

La mise en œuvre de l'outil commence généralement par un projet pilote. Le but est d'assurer que la TAS pourra être utilisée pour obtenir les avantages attendus. Les objectifs du projet pilote incluent :
- Apprendre plus de détails sur la TAS.
- Voir comment la TAS s'inscrit dans les processus, les procédures et les outils existants; identifier comment ceux-ci pourraient devoir être modifiés (Il est généralement préférable de modifier la TAS afin de l'adapter aux processus / procédures existants. Si ceux-ci devaient être adaptés pour supporter la TAS, ce devrait être au moins par une amélioration des processus eux-mêmes).
- Concevoir l'interface du testeur pour l'automatisation des tests.
- Décider des moyens standards d'utilisation, de gestion, de stockage et de maintenance de la TAS et des actifs de test, y compris l'intégration avec la gestion de la configuration et la gestion du changement (p.ex., décider des conventions de nommage des fichiers et des tests, la création de bibliothèques et définir la modularité des suites de test).
- Déterminer les métriques et les méthodes de mesure pour surveiller l'automatisation des tests en cours d'utilisation, dont la facilité d'utilisation, la maintenabilité et l'évolutivité.
- Évaluer si les bénéfices peuvent être obtenus à un cout raisonnable. Ce sera l'occasion pour remettre à zéro les attentes une fois que les TASs auront été utilisées.
- Déterminer les compétences requises et évaluer celles qui sont présentes et celles qui manquent

Identifier un projet adapté
Le projet pilote devrait être sélectionné en suivant attentivement les directives suivantes :
- Ne pas sélectionner un projet critique. Les retards subis par le déploiement de la TAS ne devraient pas avoir un impact majeur sur les projets critiques. Le déploiement de la TAS prendra du temps au début. L'équipe projet devra en être consciente.
- Ne pas sélectionner un projet trivial. Un projet trivial n'est pas un bon choix car le succès de son déploiement ne signifie pas que le déploiement des projets non triviaux se fera avec succès, et

donc apporte moins d'information utile au déploiement. Impliquer les parties prenantes nécessaires (y compris le management) dans le processus de sélection.

- Le SUT du projet pilote devrait être une bonne référence pour les autres projets de l'organisation, par exemple, le SUT devrait contenir des composants IHM représentatifs qui doivent être automatisés.

Planifier le pilote
Le pilote doit être traité comme un projet de développement normal : faire un plan, prévoir le budget et les ressources, faire le reporting d'avancement, définir les jalons, etc. Un point d'attention supplémentaire est de s'assurer que les personnes travaillant sur le déploiement de la TAS (c. à d., un champion) peuvent fournir la charge nécessaire au déploiement, même si d'autres projets exigent des ressources pour leurs propres activités. Il est important d'avoir l'engagement du management, en particulier sur les ressources partagées. Ces personnes ne seront probablement pas en mesure de travailler à temps plein sur le déploiement.

Lorsque la TAS n'a pas été fournie par un fournisseur mais est développée en interne, ses développeurs doivent être impliqués dans les activités de déploiement.

Conduire le pilote
Prêter attention aux points suivants en réalisant le déploiement du pilote :

- La TAS fournit-elle la fonctionnalité prévue (et promise par le vendeur)? Si non, cela doit être abordé dès que possible. Lorsque la TAS est développée en interne ses développeurs doivent aider au déploiement en fournissant les fonctionnalités manquantes.
- Est-ce que la TAS et le processus existant se supportent l'un et l'autre ? Dans le cas contraire ils doivent être alignés.

Évaluer le pilote
Utilisez toutes les parties prenantes pour l'évaluation.

4.1.2 Déploiement

Une fois que le pilote a été évalué, la TAS ne devrait être déployée dans le reste du département / organisation, que si le pilote est réputé réussi. Le déploiement devrait être entrepris progressivement et être bien géré. Les facteurs de réussite pour le déploiement comprennent :

- Un déploiement incrémental : Appliquer le déploiement au reste de l'organisation par étapes, par incrémentations. De cette façon, le support aux nouveaux utilisateurs se fait par «vagues» et pas en une seule fois. Cela permet d'étendre l'utilisation des TAS par étapes. De possibles goulets d'étranglement peuvent être identifiés et résolus avant qu'ils ne deviennent des problèmes réels. Des licences peuvent être ajoutées si nécessaire.
- Adapter et améliorer les processus pour qu'ils s'adaptent à l'utilisation de la TAS : Lorsque différents utilisateurs utilisent la TAS, différents processus viennent en contact avec la TAS et doivent être accordés à la TAS, ou la TAS peut nécessiter des (petites) adaptations au processus.
- Former et coacher / mentorer les nouveaux utilisateurs : les nouveaux utilisateurs ont besoin de formation et de coaching pour utiliser les nouvelles TAS. Assurez-vous que cela est en place. Des formations / ateliers devraient être donnés aux utilisateurs avant que ceux-ci utilisent effectivement les TAS.
- Définir les lignes directrices d'utilisation : Il est possible d'écrire des lignes directrices, des checklists et des FAQ pour l'utilisation de la TAS. Cela peut éviter de nombreuses questions au support.

- Mettre en œuvre un moyen de recueillir des informations sur l'utilisation réelle : Il devrait y avoir un moyen automatisé pour recueillir des informations sur l'utilisation réelle de la TAS. Idéalement, non seulement sur utilisation elle-même, mais aussi sur les parties des TAS (certaines fonctionnalités) qui sont utilisées. De cette manière, l'utilisation des TAS peut être contrôlée facilement.
- Surveiller l'utilisation de la TAS, les avantages et couts : le contrôle de l'utilisation de la TAS sur une période de temps donnée indique si la TAS est effectivement utilisée. Cette information peut également être utilisée pour recalculer le bénéfice métier (p.ex. combien de temps a été gagné, nombre de problèmes évités).
- Fournir un support aux équipes de test pour une TAS donnée.
- Collecter les leçons apprises de toutes les équipes : Effectuer une évaluation / rétrospective avec les différentes équipes qui utilisent la TAS. Cela permet d'identifier les leçons apprises. Les équipes auront le sentiment que leur contribution est utile et voudront améliorer l'utilisation de la TAS.
- Identifier et mettre en œuvre des améliorations : Identifier et mettre en œuvre les étapes pour l'amélioration en se basant sur les évaluations de l'équipe et le suivi des TAS. Il est également nécessaire de communiquer clairement avec les parties prenantes

4.1.3 Déploiement de la TAS lié au cycle de vie du développement

Le déploiement d'une TAS dépend fortement de la phase de développement du projet logiciel qui sera testé par la TAS .

Habituellement, une nouvelle TAS ou une nouvelle version de celle-ci, est déployée soit au début du projet, soit lorsqu'il atteint une étape importante, par exemple un gel de code ou la fin d'un sprint. C'est parce que les activités de déploiement, avec tous les tests et les modifications à faire, ont besoin de temps et d'efforts. C'est ainsi un bon moyen pour atténuer le risque de non fonctionnement d'une TAS qui provoquerait des perturbations dans le processus d'automatisation des tests. Cependant, si des questions essentielles se posent concernant la TAS ou si un composant de l'environnement dans lequel il s'exécute doit être remplacé, le déploiement se fera indépendamment de la phase de développement du SUT.

4.2 Évaluation des risques et stratégies d'atténuation

Des problèmes techniques peuvent mener à des risques produit ou projet. Les problèmes techniques typiques incluent :

- Trop d'abstraction peut conduire à des difficultés à comprendre ce qui se passe réellement (p.ex., avec des mots clés).
- Pilotage par les données: les tableaux de données peuvent devenir trop grands / complexes / lourds
- Dépendance de la TAS à utiliser certains composants ou DLL du système d'exploitation qui peuvent ne pas être disponibles dans tous les environnements cibles du SUT

Les risques typiques de déploiement du projet incluent :

- Problèmes de ressources: trouver les bonnes personnes pour maintenir la base de scripts peut être difficile
- Livrables du nouveau SUT qui peuvent provoquer un mauvais fonctionnement de la TAS
- Retards dans l'introduction de l'automatisation
- Retards dans la mise à jour de la TAS au regard des changements effectués sur le SUT
- La TAS ne peut pas capturer les objets (non-standards) qu'elle est supposée suivre

Les points de défaillance potentiels du projet de la TAS comprennent :

- Migration vers un environnement différent
- Déploiement de l'environnement cible
- Nouvelle livraison du développement

Un certain nombre de stratégies d'atténuation des risques peuvent être utilisées pour faire face à ces zones à risque. Celles-ci seront présentées ci-après.

La TAS a son propre cycle de vie, qu'elle soit développée en interne ou qu'il s'agisse d'une solution acquise. Une chose à retenir est que la TAS, comme tout autre logiciel, doit être sous contrôle de version et que ses fonctionnalités doivent être documentées. Sinon, il devient très difficile de déployer différentes parties de celle-ci et de les faire travailler ensemble ou dans certains environnements.

En outre, la TAS doit être documentée, claire et sa procédure de déploiement doit être facile de suivre. Cette procédure dépend de la version; par conséquent, elle doit être inclue dans la version sous contrôle.

Il y a deux cas distincts lors du déploiement d'une TAS :
1. La TAS est déployée pour la première fois
2. Déploiement de maintenance - La TAS existe déjà et doit être maintenue

Avant de commencer le premier déploiement d'une TAS, il est important de s'assurer qu'elle peut fonctionner dans son propre environnement, qu'elle est isolée des changements aléatoires, et que les cas de test peuvent être mis à jour et gérés. La TAS ainsi que son infrastructure doivent être maintenues .

Dans le cas du premier déploiement, les étapes de base qui suivent sont nécessaires :
- Définir l'infrastructure dans laquelle la TAS fonctionnera
- Créer l'infrastructure pour la TAS
- Créer une procédure pour maintenir la TAS et son infrastructure
- Créer une procédure pour maintenir la suite de test que la TAS exécutera

Les risques relatifs au premier déploiement comprennent :
- Le temps total d'exécution de la suite de test peut être plus long que le temps d'exécution prévu pour le cycle de test. Dans ce cas, il est important de veiller à ce que la suite de test dispose d'assez de temps pour être exécutée entièrement avant que ne commence le prochain cycle de test prévu .
- Il existe des problèmes d'installation et de configuration avec l'environnement de test (par exemple, configuration de base de données et charge initiale, démarrage/arrêt des services). En général, la TAS a besoin d'un moyen efficace pour configurer les conditions préalables requises pour les cas de test automatisés dans l'environnement de test.

Pour les déploiements suivants, des considérations supplémentaires sont à prendre en compte. La TAS elle-même a besoin d'évoluer, et les mises à jour nécessaires pour cela devront être déployées en production. Avant de déployer une mise à jour de la TAS en production, elle devra être testée comme tout autre logiciel. Il est donc nécessaire de vérifier les nouvelles fonctionnalités, pour s'assurer que la suite de test pourra être exécutée avec la TAS mise à jour, que les rapports pourront être envoyés, et qu'il n'y aura pas de problèmes de performance ou d'autres régressions fonctionnelles. Dans certains cas, l'ensemble de la suite de test peut nécessiter d'être modifié afin de s'adapter à la nouvelle version de la TAS .

Lorsque cela se produit, les étapes suivantes sont nécessaires :
- Évaluer les évolutions de la nouvelle version de la TAS par rapport à l'ancienne

- Tester les nouvelles fonctionnalités de la TAS et les régressions
- Vérifier si la suite de test doit être adaptée à la nouvelle version de la TAS

Une mise à jour entraine également les risques suivants :

- La suite de test doit être modifiée afin de fonctionner avec la mise à jour de la TAS : Modifier alors la suite de test et la tester avant de la déployer sur la TAS.
- Les bouchons, les pilotes et les interfaces utilisés pour tester doivent être modifiés pour s'adapter à la mise à jour de la TAS : faire les changements nécessaires au harnais de test et le tester avant de déployer la TAS .
- L'infrastructure doit changer pour tenir compte de la mise à jour de la TAS : évaluer les composants d'infrastructure qui doivent être changés, effectuer les modifications et les tester avant de mettre à jour la TAS.
- Les mises à jour de la TAS ont une performance moindre, des défauts ou des régressions: évaluer les risques par rapport aux bénéfices. Si les problèmes découverts rendent impossible la mise à jour de la TAS, il peut être préférable de ne pas procéder à la mise à jour ou d'attendre une prochaine version de la TAS. Si les problèmes soulevés sont négligeables par rapport aux avantages, la TAS peut encore être mise à jour. N'oubliez pas de créer un bordereau de livraison avec les problèmes connus pour informer les ingénieurs d'automatisation de test et les autres parties prenantes et essayer de prévoir le moment où vont être soulevées les questions.

4.3 Maintenance des tests automatisés

Développer des solutions d'automatisation de test n'est pas trivial. Elles doivent être modulaires, évolutives, compréhensibles, fiables et vérifiables. Pour ajouter encore plus de complexité, des solutions d'automatisation de test - comme n'importe quel autre système logiciel, sont appelées à évoluer. Que ce soit dû à des changements internes ou à des changements dans l'environnement dans lequel elles opèrent, la maintenance est un aspect important de l'architecture d'une TAS. Maintenir la TAS en l'adaptant aux nouveaux types de systèmes à tester, en l'accommodant à de nouveaux environnements logiciels, ou en la rendant conforme aux nouvelles lois et réglementations, contribue à assurer un fonctionnement fiable et sûr de la TAS. Cela optimise également la durée de vie et la performance de la TAS.

4.3.1 Types de Maintenance

La maintenance est effectuée sur une TAS opérationnelle existante et est déclenchée par les modifications, les migrations, ou son retrait du système. Ce processus peut être structuré suivant les catégories suivantes :

- Maintenance préventive - Les changements sont faits pour que la TAS supporte plusieurs types de tests, le test sur des interfaces multiples, le test de plusieurs versions du SUT ou permette des tests automatisés pour un nouveau SUT.
- Maintenance corrective - Des modifications sont apportées pour corriger les défaillances de la TAS. La meilleure façon de maintenir une TAS en exploitation, réduisant ainsi le risque à l'utiliser, est d'exécuter des tests réguliers de maintenance.
- Maintenance perfective - La TAS est optimisée et des problèmes non-fonctionnels sont corrigés. Ils peuvent concerner la performance de la TAS, sa facilité d'utilisation, sa robustesse ou sa fiabilité.
- Maintenance adaptative - Comme de nouveaux systèmes logiciels sont lancés sur le marché (systèmes d'exploitation, gestionnaires de bases de données, navigateurs Web etc...), il peut être nécessaire que la TAS les supporte. En outre, il est possible que la TAS doive se conformer à de

nouvelles lois, règlements ou prescriptions spécifiques à l'industrie. Dans ce cas, des modifications sont apportées à la TAS pour l'adapter en conséquence.
Remarque: Habituellement, la conformité aux lois et aux règlements entraine une maintenance obligatoire qui a ses propres règles, exigences et quelquefois des exigences d'audit. En outre, comme les outils d'intégration sont mis à jour et que des nouvelles versions sont créées, les outils d'intégration doivent être maintenus et gardés fonctionnels.

4.3.2 Portée et approche

La maintenance est un processus qui peut affecter toutes les couches et tous les composants d'une TAS. Son périmètre dépend de :

- La taille et la complexité de la TAS
- La taille du changement
- Le risque lié au changement

Compte tenu du fait que la maintenance concerne la TAS en cours d'exploitation, une analyse d'impact est nécessaire pour déterminer comment le système peut être affecté par les changements. Selon l'impact, les changements doivent être introduits progressivement et les tests doivent être effectués après chaque étape pour assurer le fonctionnement continu de la TAS. Remarque: la maintenance de la TAS peut être difficile si ses spécifications et sa documentation ne sont pas à jour.

Comme l'efficacité en temps est le principal facteur contribuant à la réussite de l'automatisation des tests, il devient essentiel d'avoir de bonnes pratiques pour maintenir la TAS y compris :

- Les procédures de déploiement et d'utilisation de la TAS doivent être claires et documentées
- Les dépendances avec des parties tierces doivent être documentées, avec les inconvénients et problèmes connus
- La TAS doit être modulaire, ainsi certaines parties peuvent être facilement remplacées
- La TAS doit fonctionner dans un environnement qui est remplaçable ou avec des composants remplaçables
- La TAS doit séparer les scripts de test du TAF lui-même
- La TAS doit s'exécuter isolément de l'environnement de développement, afin que les changements accidentels ou aléatoires n'affectent pas l'environnement de test
- La TAS avec son environnement, les suites de test et les artefacts de test doivent être sous gestion de configuration

Des considérations existent pour la maintenance des composants et des bibliothèques tierces :

- Très souvent, la TAS va utiliser des composants tiers pour exécuter les tests. La TAS peut également dépendre de librairies tierces (par exemple, les bibliothèques d'automatisation de l'interface utilisateur). Tous les composants tiers de la TAS doivent être documentés et sous gestion de configuration.
- Il est nécessaire d'avoir un plan au cas où ces composants externes devraient être modifiés ou corrigés. La personne responsable de la maintenance de la TAS a besoin de savoir à qui s'adresser ou à qui soumettre une question.
- Il doit y avoir une documentation concernant la licence sous laquelle les composants tiers sont utilisés, de sorte que l'on sache si ils peuvent être modifiés, dans quelle mesure et par qui.
- Pour chacun des composants tiers, il est nécessaire d'obtenir des informations sur les mises à jour et les nouvelles versions. Garder les composants tiers et les bibliothèques à jour est une action préventive qui rentabilise l'investissement sur le long terme.

Les considérations pour les standards de nommage et d'autres conventions comprennent :

- L'idée de standards de nommage et autres conventions repose sur une raison simple : la suite de test et la TAS elle-même doivent être faciles à lire, comprendre, modifier et maintenir. Cela économise du temps de maintenance et minimise également le risque d'introduction de régressions ou de mauvaises corrections qui pourraient être facilement évitées.
- Il est plus facile d'intégrer de nouvelles personnes dans le projet d'automatisation de test lorsque des conventions de nommage standards sont utilisées.
- Les standards de nommage peuvent se référer à des variables et des fichiers, à des scénarios de test, des mots clés et des paramètres de mots clés. D'autres conventions renvoient à des prérequis et post-actions pour l'exécution du test, au contenu des données de test, à l'environnement de test, à l'état d'exécution de test et aux journaux d'exécution et rapports.
- Tous les standards et les conventions doivent être convenus et documentés au démarrage d'un projet d'automatisation de test.

Considérations concernant la documentation à prendre en compte :

- La nécessité d'une documentation mise à jour et de qualité pour à la fois les scénarios de test et la TAS est très claire, mais cela entraine deux questions: quelqu'un doit écrire cette documentation et quelqu'un doit la maintenir à jour.
- Bien que le code de l'outil de test puisse être auto-documenté ou semi-automatiquement documenté, toute la conception, les composants, les intégrations avec les tiers, les dépendances et les procédures de déploiement doivent être documentés par quelqu'un.
- Habituellement, une bonne pratique est d'introduire l'écriture de la documentation dans le cadre du processus de développement. Une tâche qui ne devrait pas être considérée comme terminée sauf si elle est documentée ou si la documentation est mise à jour.

Considérations concernant le matériel de formation à prendre en compte :

- Si la documentation de la TAS est bien écrite, elle peut être utilisée comme base pour le matériel de formation de la TAS.
- Habituellement, le matériel de formation est une combinaison de spécifications fonctionnelles de la TAS, de la conception et de l'architecture de la TAS, du déploiement et de la maintenance de la TAS, de l'utilisation de la TAS (manuel d'utilisateur), d'exemples pratiques et d'exercices, de trucs et astuces.
- La maintenance du matériel de formation consiste d'abord à l'écrire puis à la revoir périodiquement. Cela est fait en pratique par les membres de l'équipe désignés comme formateurs de la TAS et se passe généralement vers la fin d'une itération du cycle de vie du SUT (à la fin de sprints, par exemple).

5 Métriques et reporting sur l'automatisation des tests - 165 mins.

Mots-clés

Densité de défauts de code d'automatisation, matrice de couverture, effort de test manuel équivalent, métriques, enregistrement des tests, reporting des tests

Objectifs d'apprentissage pour métriques et reporting de l'automatisation des tests

5.1 Sélection des Métriques sur la TAS

ALTA-E-5.1.1 (K2) Classer les métriques qui peuvent être utilisées pour suivre la stratégie d'automatisation des tests et son efficacité

5.2 Mise en œuvre des métriques

ALTA-E-5.2.1 (K3) Implémenter des méthodes de collecte de métriques pour prendre en charge les exigences techniques et de gestion. Expliquez comment la mesure de l'automatisation des tests peut être implémentée.

5.3 Journalisation de la TAS et du SUT

ALTA-E-5.3.1 (K4) Analyser les données de journalisation des tests pour la TAS et le SUT

5.4 Automatisation du reporting des tests

ALTA-E-5.4.1 (K2) Expliquer comment un rapport d'exécution est construit et publié

5.1 Sélection des mesures de la TAS

Cette section concerne les métriques qui peuvent être utilisées pour suivre la stratégie d'automatisation de test, l'efficacité et l'efficience de la TAS. Celles-ci sont différentes des métriques utilisées pour suivre le SUT et les tests du SUT. Ces métriques sont choisies par le Test Manager du projet. Les métriques d'automatisation de test permettent au TAM et au TAE de suivre l'avancement de l'atteinte des objectifs d'automatisation de test et de suivre l'impact des modifications apportées à la solution d'automatisation de test.

Les métriques de la TAS peuvent être divisées en deux groupes : externes et internes. Les métriques externes sont celles utilisées pour mesurer l'impact de la TAS sur d'autres activités (en particulier les activités de test). Les métriques internes sont celles utilisées pour mesurer l'efficacité et l'efficience de la TAS dans l'accomplissement de ses objectifs.

Les métriques de la TAS sont généralement les suivantes :

Métriques externes de la TAS
- Avantages de l'automatisation
- Effort de construction des tests automatisés
- Effort d'analyse des incidents de tests automatisés
- Effort de maintenance des tests automatisés
- Ratio de pannes/défauts
- Temps d'exécution des tests automatisés
- Nombre de cas de tests automatisés
- Nombre de résultats réussis ou en échecs
- Nombre de faux-échecs et de faux succès
- Couverture de code

Métriques internes de la TAS
- Métriques d'outils de script
- Densité de code défectueux d'automatisation
- Vitesse et efficacité des composants TAS

Celles-ci sont décrites ci-dessous.

Avantages de l'automatisation
Il est particulièrement important de mesurer et de présenter les avantages de la TAS. Cela parce-que les couts (en terme de nombre de personnes concernées sur une période de temps donnée) sont faciles à voir. Les personnes qui travaillent en dehors des tests pourront se faire une idée du cout global, mais elles ne peuvent pas voir les avantages obtenus.

Toute mesure de bénéfice dépendra de l'objectif de la TAS. Généralement, cela peut être une économie de temps ou d'effort, une augmentation de la quantité de tests effectués (la largeur ou la profondeur de la couverture, ou la fréquence d'exécution), ou d'autres avantages comme une répétabilité accrue, une plus grande utilisation des ressources, ou moins d'erreurs manuelles.

Les métriques possibles comprennent :
- Nombre d'heures d'effort de test manuel économisées

- Réduction du temps d'exécution des tests de régression
- Nombre de cycles supplémentaires d'exécution de test réalisés
- Nombre ou pourcentage de tests supplémentaires exécutés
- Pourcentage de cas de test automatisés par rapport à l'ensemble des cas de test (bien que les tests ne puissent pas facilement être comparés aux cas de test manuels)
- Augmentation de la couverture (exigences, fonctionnalités, structurelle)
- Nombre de défauts trouvés plus tôt grâce à la TAS (quand le bénéfice moyen de défauts trouvés tôt est connu, cela peut être "calculé" comme une somme des couts évités)
- Nombre de défauts constatés grâce à la TAS qui n'auraient pas été trouvés par des tests manuels (par exemple, les défauts de fiabilité)

Notez que l'automatisation des tests économise généralement de l'effort de test manuel. Cet effort peut être consacré à d'autres types de tests (manuels) (par exemple, les tests exploratoires). Les défauts trouvés par ces tests supplémentaires peuvent également être vus comme des avantages indirects de la TAS, puisque l'automatisation des tests permet à ces tests manuels d'être exécutés. Sans la TAS ces tests n'auraient pas été exécutés et par la suite les défauts supplémentaires n'auraient pas été trouvés.

Effort de construction des tests automatisés
L'effort d'automatisation des tests est l'un des couts principaux associés à l'automatisation de test. Il est souvent plus important que le cout d'exécution du même test manuellement et donc il peut être un frein à l'expansion de l'utilisation de l'automatisation de test. Alors que le cout de mise en œuvre d'un test automatisé spécifique dépend en grande partie du test lui-même, d'autres facteurs tels que l'approche de script utilisée, la familiarité avec l'outil de test, l'environnement, et le niveau de compétences de l'ingénieur d'automatisation des tests auront également un impact.

Parce que des tests grands ou complexes prennent généralement plus de temps à être automatisés que des tests courts ou simples, le calcul du cout de fabrication de l'automatisation des tests peut être basé sur un temps de fabrication moyen. Ceci peut être encore affiné en tenant compte du cout moyen pour une série de tests spécifiques tels que ceux qui ciblent la même fonction ou ceux concernant un niveau de test donné. Une autre approche consiste à exprimer le cout de construction comme un facteur de l'effort requis pour exécuter le test manuellement (effort de test manuel équivalent, EMTE). Par exemple, l'automatisation d'un test peut être de deux fois l'effort de test manuel, ou deux fois le EMTE.

Effort pour analyser les incidents du SUT
L'analyse des échecs dans le SUT découverts par l'exécution de test automatisée peut être beaucoup plus complexe que pour un test exécuté manuellement car les événements menant à la défaillance d'un test manuel sont souvent connus par le testeur exécutant le test. Cela peut être atténué comme décrit au niveau de la conception au chapitre 3.1.4 et au niveau des rapports dans les chapitres 5,3 et 5,4. Cette mesure peut être exprimée comme une moyenne par cas d'échec de test ou elle peut être exprimée en tant que facteur d'EMTE. Ce dernier étant particulièrement approprié lorsque les tests automatisés varient considérablement en complexité et en longueur d'exécution. Les enregistrements disponibles du SUT et de la TAS joue un rôle crucial dans l'analyse des échecs. La journalisation doit fournir suffisamment d'informations pour effectuer cette analyse de manière efficace. Les fonctions d'enregistrement importantes incluent:

- L'enregistrement du SUT et l'enregistrement de la TAS doivent être synchronisés
- Le TAS doit enregistrer le comportement attendu et réel
- Le TAS doit enregistrer les actions à effectuer

Le SUT, d'un autre côté, devrait enregistrer toutes les actions qui sont effectuées (sans savoir si l'action est le résultat d'un test manuel ou automatisé). Toutes les erreurs internes doivent être enregistrées et tous les vidages mémoire et les traces de pile devraient être disponibles.

Effort pour maintenir les tests automatisés
L'effort de maintenance requis pour garder les tests automatisés synchronisés avec le SUT peut être très important et, finalement, être plus important que les avantages obtenus par la TAS. Cela a été la cause de l'échec de nombreux efforts d'automatisation. Le suivi de l'effort de maintenance est donc important à mettre en évidence lorsque des mesures doivent être prises pour réduire l'effort de maintenance ou au moins pour l'empêcher de se développer sans contrôle.

Les métriques de l'effort de maintenance peuvent être exprimées de façon globale pour tous les tests automatisés nécessitant une mise à jour à chaque nouvelle version du SUT. Elles peuvent également être exprimées en moyenne par test automatisé mis à jour ou comme un facteur de EMTE.

Une métrique connexe est le nombre ou le pourcentage de tests nécessitant d'être maintenus.

Lorsque l'effort de maintenance pour l'automatisation des tests est connu (ou peut être déduit), cette information peut jouer un rôle déterminant pour décider si oui ou non il faut mettre en œuvre certaines fonctionnalités ou corriger certains défauts.
L'effort requis pour maintenir les cas de test à cause de modifications du logiciel, doit être envisagé avec la modification du SUT.

Ratio de pannes/ défauts
Un problème courant avec les tests automatisés est que beaucoup d'entre eux peuvent échouer pour la même raison - un seul défaut dans le logiciel. Alors que le but des tests est de mettre en évidence des défauts dans le logiciel, avoir plus d'un test en surbrillance présentant le même défaut est une perte de temps. Ceci est particulièrement le cas pour les tests automatisés dont l'effort nécessaire d'analyse de chaque test échoué peut être important. Mesurer le nombre de tests automatisés qui échouent pour un défaut donné peut aider à indiquer qu'il y a peut-être un problème. La solution réside dans la conception des tests automatisés et leur sélection pour exécution.

Temps d'exécution des tests automatisés
L'une des métriques les plus faciles à déterminer est le temps nécessaire pour exécuter les tests automatisés. Au début de la TAS cela pourrait ne pas être important, mais quand le nombre de cas de test automatisés augmente, cette métrique peut devenir très importante.

Nombre de cas de tests automatisés
Cette métrique peut être utilisée pour montrer l'avancement du projet d'automatisation de test. Mais il faut tenir compte du fait que le nombre de cas de tests automatisés ne révèle pas beaucoup d'informations à lui tout seul ; par exemple, il n'indique pas que la couverture de test a augmenté.

Nombre de résultats en échec ou en succès
Cette métrique courante révèle combien de tests automatisés ont été passés et combien ont échoué. Les échecs doivent être analysés afin de déterminer si l'échec est dû à un défaut dans la TAS, ou est dû à des problèmes externes comme un problème d'environnement, ou à la TAS elle-même.

Nombre de résultats de faux-échecs et de faux-succès
Comme on l'a vu dans plusieurs métriques précédentes, l'analyse des échecs peut prendre un certain temps à analyser. C'est encore plus frustrant quand l'échec s'avère être une fausse alarme. Cela se produit

lorsque le problème est dans la TAS ou dans le cas de test, mais pas dans le SUT. Il est important que le nombre de fausses alarmes (et l'effort potentiellement gaspillé) soit maintenu à un bas niveau. Les faux-échecs peuvent réduire la confiance dans la TAS. Inversement, les résultats de faux-succès peuvent être plus dangereux. Quand un faux-succès se produit, c'est qu'il y avait une défaillance dans le SUT, mais qu'elle n'a pas été identifiée par l'automatisation des tests de sorte qu'un résultat de faux-succès a été signalé. Dans ce cas, un défaut potentiel peut échapper à la détection. Cela peut se produire parce que la vérification du résultat n'a pas été faite correctement, qu'un oracle de test non valide a été utilisé ou que le résultat du cas de test était erroné.

Notez que les fausses alarmes peuvent être causées par des défauts dans le code de test (voir "métrique de densité de défauts de code d'automatisation"), mais peuvent aussi être causées par l'instabilité du SUT qui se comporte de façon imprévisible (p.ex., temps dépassés). Des Hook de test peuvent également causer de fausses alarmes en raison de leur niveau d'intrusion.

Couverture de code
La connaissance de la couverture de code fournie par les différents cas de test peut révéler des informations utiles. Cela peut aussi être mesuré à un niveau élevé, par exemple, au niveau de la couverture de code de la suite de test de régression. Il n'y a pas pourcentage absolu qui indique une couverture adéquate, et une couverture de code à 100% est impossible à atteindre autrement que dans la plus simple des applications logicielle. Cependant, il est généralement admis que plus la couverture est bonne, plus le risque global de déploiement de logiciels est réduit. Ainsi, cette métrique peut indiquer une activité dans le SUT. Par exemple, si la couverture de code diminue, cela signifie très probablement que la fonctionnalité a été ajoutée au SUT, mais qu'aucun cas de test correspondant n'a été ajouté à la suite de tests automatisés.

Métriques d'outils de scripting
Beaucoup de métriques peuvent être utilisées pour suivre le développement de scripts d'automatisation. La plupart d'entre elles sont semblables aux métriques utilisées sur le code source du SUT. Les lignes de code (LOC) et la complexité cyclomatique peuvent être utilisées pour mettre en évidence des scripts trop importants ou trop complexes (Suggérant la nécessité d'une reconception).

Le ratio de commentaires par instruction exécutable peut être utilisé pour donner une indication de l'étendue de la documentation et des annotations du script. Le nombre de non-conformités aux normes de scripting peut donner une indication du niveau de respect de ces normes.

Densité de défaut de code d'automatisation
Le code d'automatisation n'est pas différent du code du SUT car il s'agit d'un logiciel, et il peut contenir des défauts. Il ne doit pas être considéré comme moins important que le code du SUT. De bonnes pratiques de codage et l'application de normes devraient être mises en œuvre, et leurs résultats suivis par des métriques telles que la densité de défauts de code. Ceux-ci seront plus faciles à collecter avec l'aide d'un système de gestion de configuration.

Vitesse et efficacité des composants de la TAS
Les différences de durées pour réaliser de mêmes étapes de test dans un même environnement peuvent indiquer un problème dans le SUT. Investiguer est nécessaire quand le SUT n'effectue pas les mêmes fonctionnalités dans le même temps. Cela peut indiquer une variabilité dans le système, ce qui n'est pas acceptable et qui pourrait s'aggraver avec une augmentation de la charge. En outre, la TAS doit être aussi efficace que le SUT, de sorte que sa performance n'affecte pas la performance du SUT pendant l'exécution du test.

Indicateurs de tendance
Pour beaucoup de ces mesures, ce sont les tendances (à savoir la façon dont les mesures changent avec le temps) qui apportent plus que la valeur d'une métrique à un instant T. Par exemple, savoir que le cout moyen de maintenance pour les tests automatisés nécessitant une maintenance est plus élevé que ce qu'il était pour les deux versions précédentes du SUT, peut inciter à déterminer la cause de l'augmentation de ce cout et à réagir pour inverser la tendance.

Le cout de mesure doit être aussi faible que possible, ce qui peut souvent être atteint en automatisant la collecte et le reporting.

5.2 Mise en œuvre de la mesure

Puisqu'une stratégie d'automatisation de test est basée sur un logiciel de test automatisé, ce logiciel automatisé peut être amélioré pour enregistrer des informations sur son utilisation. Lorsque l'abstraction est combinée avec un logiciel structuré, toutes les améliorations apportées au logiciel de test peuvent être utilisées par tous les scripts de test automatisés de niveau supérieur. Par exemple, l'amélioration du logiciel pour qu'il enregistre l'heure de début et de fin d'exécution d'un test peut s'appliquer à tous les tests.

Caractéristiques de l'automatisation permettant les mesures et la génération de rapports
Les langages de script de nombreux outils de test acceptent des métriques et génèrent des rapports par le biais d'aménagements qui peuvent être utilisés pour enregistrer les informations avant, pendant et après l'exécution de tests individuels, d'ensembles de tests et de suite de test complètes.
Le reporting de chacune des séries de tests exécutées doivt avoir une fonction d'analyse pour prendre en compte les résultats de test précédemment exécutés, de façon à ce qu'il mette en évidence les tendances (tels que les changements dans le taux de réussite de test).

Automatiser les tests nécessite généralement l'automatisation à la fois de l'exécution du test et de sa vérification, ce dernier étant obtenu en comparant des éléments spécifiques du résultat du test avec des résultats attendus prédéfinis. Cette comparaison est généralement mieux faite par un outil de test. Le niveau d'information rapporté en tant que résultat de cette comparaison doit être pris en compte. Il est important que le statut du test soit déterminé correctement. (p.ex., réussite, échec). Dans le cas d'un statut d'échec, il sera nécessaire d'obtenir plus d'informations sur sa cause (p.ex., copies d'écran).

Distinguer les différences dans le résultat réel et attendu d'un test n'est pas toujours trivial bien que les outils de support puissent aider grandement pour définir des comparaisons qui ignorent des différences attendues (comme les dates et les heures), tout en mettant en évidence les différences inattendues.

Intégration avec d'autres outils tiers (feuilles de calcul, XML, documents, bases de données, outils de rapport, etc.)
Lorsque les informations de l'exécution du cas de test automatisé sont utilisées dans d'autres outils (pour le suivi et la communication), il est possible de fournir les informations dans un format qui convienne à ces outils tiers. Cela est souvent obtenu grâce aux fonctionnalités existantes de l'outil de test (formats d'exportation pour les rapports) ou en créant des rapports personnalisés qui sont générés dans un format compatible avec les autres programmes (".xls" pour Excel, «.doc» pour Word, ".html" pour Web, etc.)

Visualisation des résultats (tableaux de bord, tableaux, graphiques, etc.)
Les résultats des tests doivent être rendus visibles dans des graphiques. Envisagez d'utiliser des couleurs pour indiquer des problèmes dans l'exécution des tests tels que des feux de circulation pour indiquer l'état

d'avancement de l'exécution du test / de l'automatisation afin que les décisions puissent être prises sur la base des informations transmises. Le management est particulièrement intéressé par des résumés visuels permettant de voir le résultat du test d'un coup d'œil ; Au cas où de plus amples informations seraient nécessaires, il sera possible de plonger dans les détails.

5.3 Enregistrement de la TAS et du SUT

L'enregistrement est très important dans la TAS, comprenant l'enregistrement à la fois pour l'automatisation de test elle-même et pour le SUT. Les enregistrements de test sont une source fréquemment utilisée pour analyser les problèmes potentiels. Vous trouverez dans la section suivante des exemples d'enregistrements de test classés par TAS ou SUT.

L'enregistrement de la TAS doit inclure les points suivants (que le TAF ou le cas de test lui-même enregistre les informations n'est pas si important que cela et dépend du contexte) *:*

- Le cas de test actuellement en cours d'exécution, y compris l'heure de début et de fin.
- L'état de l'exécution du cas de test parce que, tandis que les défaillances peuvent être facilement identifiées dans les fichiers d'enregistrement, le Framework devrait également avoir cette information et la signaler via un tableau de bord. L'état d'exécution du cas de test peut être Réussi, Echec ou Erreur. Le résultat de l'erreur est utilisé dans des situations où il n'est pas clair que l'erreur provienne du SUT.

- Les détails de l'enregistrement de test à un niveau élevé (enregistrement des étapes significatives), y compris des informations temporelles.
- Les informations dynamiques sur le SUT (par exemple, les fuites de mémoire) que le cas de test a été en mesure d'identifier à l'aide d'outils tiers. Les résultats et les défaillances de ces mesures dynamiques réelles doivent être enregistrés avec le cas de test qui s'exécutait lorsque l'incident a été détecté.
- Dans le cas de test de fiabilité / tests de stress (où de nombreux cycles sont effectués) un compteur doit être enregistré, de sorte qu'il puisse être facilement déterminé combien de fois des cas de test ont été exécutés.
- Lorsque des cas de test ont des parties aléatoires (par exemple, des paramètres aléatoires, ou des étapes aléatoires dans les tests d'état machine), les nombres / choix aléatoires doivent être enregistrés.
- Toutes les actions effectuées par un cas de test devraient être enregistrées de manière à ce que le fichier d'enregistrement (ou des parties de celui-ci) puisse être relu afin de ré-exécuter le test avec exactement les mêmes étapes et le même timing. Cela est utile pour vérifier la reproductibilité d'une défaillance identifiée et capturer des informations supplémentaires. L'information sur l'action de cas de test pourrait également être enregistrée sur le SUT pour être utilisée lors de la reproduction des problèmes identifiés par le client (le client exécute le scénario, les informations de l'enregistrement sont capturées et il peut alors être rejoué par l'équipe de développement lors de la résolution du problème).
- Les captures d'écran et autres captures visuelles peuvent être sauvegardées pendant l'exécution du test pour une utilisation ultérieure lors de l'analyse de défaillance
- Chaque fois qu'un cas de test rencontre une défaillance, la TAS doit s'assurer que toutes les informations nécessaires pour analyser le problème sont disponibles / stockées, ainsi que, le cas échéant, toute information concernant la poursuite des tests. Tout vidage mémoire associé et les traces de la pile doivent être enregistrés par la TAS dans un endroit sûr. Aussi, tous les fichiers d'enregistrement qui pourraient être écrasés (les buffers cycliques sont souvent utilisés pour les

fichiers d'enregistrement du SUT) doivent être copiés à un endroit où ils seront disponibles pour une analyse ultérieure.
- L'utilisation de la couleur peut aider à distinguer différents types d'informations journalisées (par exemple, les erreurs en rouge, les informations de progression en vert).

Enregistrement du SUT:
- Lorsque le SUT identifie un problème, toutes les informations nécessaires pour l'analyser devraient être connectées, y compris l'horodatage - date et heure, l'emplacement de la source du problème, les messages d'erreur, etc.
- Le SUT peut enregistrer toutes les interactions de l'utilisateur (directement via l'interface utilisateur disponible, mais aussi via des interfaces de réseau, etc.). De cette façon, les problèmes identifiés par les clients peuvent être analysés correctement, et l'équipe de développement peut essayer de reproduire le problème.
- Au démarrage du système, les informations de configuration doivent être enregistrées dans un fichier, composé des différentes versions de logiciels / micrologiciels, de la configuration du SUT, de la configuration de l'OS, etc.

Toutes les différentes informations d'enregistrement doivent être facilement consultables. Un problème identifié dans le journal d'enregistrement par la TAS devrait être facilement identifié dans le journal d'enregistrement du SUT, et vice-versa (avec ou sans outillage supplémentaire). La synchronisation des différents journaux avec un horodatage facilite la corrélation de ce qui s'est passé quand une erreur a été signalée.

5.4 Reporting de l'automatisation de tests

Les enregistrements de test donnent des informations détaillées sur l'exécution des étapes, des actions et des réponses d'un cas de test et / ou d'une suite de test. Cependant, les enregistrements seuls ne peuvent pas donner un bon aperçu du résultat global d'exécution. Pour cela, il est nécessaire d'avoir des fonctionnalités de reporting. Après chaque exécution de la suite de test, un rapport concis doit être créé et publié. Un composant générateur de rapports réutilisables peut être utilisé à cette fin.

Contenu des rapports
Le rapport d'exécution de test doit contenir un résumé donnant un aperçu des résultats de l'exécution, le système testé et l'environnement dans lequel les tests ont été exécutés. La capacité de détailler davantage les rapports de test n'est pas nécessaire ; Cependant, il est utile d'avoir la possibilité de rechercher dans les rapports.

A part cela, il est nécessaire de savoir quels tests ont échoué et les raisons de ces échecs. Pour faciliter le dépannage, il est important de connaître l'historique de l'exécution du test et qui est son responsable (généralement la personne qui a l'a créé ou mis à jour). La personne responsable doit enquêter sur la cause de la défaillance, signaler les problèmes liés, suivre la correction du(es) problème(s), et vérifier que le correctif a été correctement mis en œuvre.

Le reporting est également utilisé pour diagnostiquer les défaillances des composants du TAF (voir Chapitre 7).

Publication des rapports
Le rapport doit être publié pour toute personne intéressée par les résultats de l'exécution. Il peut être téléchargé sur un site Web, envoyé à une liste de diffusion ou envoyé à un autre outil comme un outil de

gestion de test. D'un côté pratique, il est plus que probable que ceux intéressés par le résultat de l'exécution le regarderont et l'analyseront si on leur donne des facilités de souscription et s'ils peuvent recevoir le rapport par email.

Facultativement, pour identifier des parties problématiques du SUT, il s'agit de conserver un historique des rapports, de sorte que des statistiques sur des cas de test ou des groupes de tests avec des régressions fréquentes puissent être recueillies.

6 Transition du test manuel vers un environnement automatisé - 120 mins.

Mots clés
re-test, tests de régression

Objectifs d'apprentissage pour la transition du test manuel vers un environnement automatisé

6.1 Critères d'automatisation

ALTA-E-6.1.1 (K3) Analyser les facteurs pertinents pour déterminer les critères appropriés à l'automatisation des tests
ALTA-E-6.1.2 (K2) Comprendre les facteurs de transition entre le test manuel et l'automatisation

6.2 Identifier les étapes nécessaires pour automatiser les tests de régression

ALTA-E-6.2.1 (K2) Expliquer les étapes nécessaires pour automatiser les tests de régression

6.3 Facteurs à considérer pour l'automatisation des tests d'une nouvelle fonctionnalité

ALTA-E-6.3.1 (K2) Expliquer les facteurs à considérer pour automatiser les tests d'une nouvelle fonctionnalité

6.4 Facteurs à considérer pour l'automatisation des re-tests des défauts

ALTA-E-6.4.1 (K2) Expliquer les facteurs à considérer pour l'automatisation des re-tests des défauts

6.1 Critères d'automatisation

Les organisations développent traditionnellement des cas de test manuels. Au moment de décider de migrer vers un environnement de test automatisé, il faut évaluer l'état actuel des tests manuels et déterminer l'approche la plus efficace pour la transition de ce référentiel de test vers le nouveau paradigme de l'automatisation. La structure existante d'un test manuel peut ou peut ne pas être adaptée à l'automatisation, dans ce cas, une réécriture complète du test pour le rendre automatisable peut être nécessaire. Ou alternativement, les éléments de tests manuels existants pertinents (par exemple, les valeurs d'entrées, les résultats attendus, le chemin de navigation) peuvent être extraits des tests manuels existants et être réutilisés pour l'automatisation. Une stratégie de test manuel qui prend en compte l'automatisation permettra des tests dont la structure facilite la migration vers l'automatisation.

Tous les tests ne peuvent ou ne doivent pas être automatisés, et parfois la première itération d'un test peut être manuelle. Il y a donc deux aspects concernant la transition à envisager : la conversion initiale de tests manuels existants en tests automatisés et par la suite l'automatisation de nouveaux tests manuels.

Notez également que certains types de test ne peuvent être exécutés (efficacement) que de façon automatisée, par exemple les tests de fiabilité, les tests de stress ou les tests de performance.

Avec l'automatisation des tests, il est possible de tester les applications et les systèmes sans interface utilisateur. Dans ce cas, le test peut être fait au niveau intégration via des interfaces dans le logiciel.
Bien que ces types de cas de test puissent également être exécutés manuellement (en entrant manuellement des commandes pour déclencher les interfaces), cela peut ne pas être pratique. Grâce à l'automatisation, il est également possible d'insérer des messages dans le système de file de messages. De cette façon, les tests peuvent commencer plus tôt (et l'on peut identifier les défauts plus tôt), lorsque le test manuel n'est pas encore possible .

Avant de commencer l'automatisation de test, il faut considérer la faisabilité et la viabilité des tests automatisés par rapport aux tests manuels.
Les critères ci-dessous sont à prendre en compte, mais pas seulement :

- Fréquence d'utilisation
- Complexité à automatiser
- Compatibilité avec des outils
- Maturité du processus de test
- Adéquation de l'automatisation avec l'étape du cycle de vie du produit logiciel
- Durabilité de l'environnement automatisé
- Possibilité de contrôle du SUT

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

Fréquence d'utilisation
Le nombre de fois qu'un test doit être exécuté est à prendre en considération pour automatiser ou pas. Les tests qui sont exécutés régulièrement, dans le cadre d'un cycle majeur ou mineur, sont de meilleurs "candidats" pour l'automatisation car ils seront utilisés fréquemment. En règle générale, plus le nombre de releases est élevé - et par conséquent les cycles de test - plus l'intérêt d'automatiser les tests est grand.

Comme les tests fonctionnels sont automatisés, ils peuvent être utilisés dans les versions ultérieures pour les tests de régression. Les tests automatisés utilisés pour les tests de régression auront un fort retour sur investissement (ROI) et atténueront les risques pour la base de code existante.

Si un script de test n'est exécuté qu'une fois par an, et si le SUT ne change qu'une fois dans l'année, il peut ne pas être possible ou pertinent de créer un test automatisé. Le temps passé à adapter un test pour un changement annuel de la SUT pourrait être mieux utilisé en testant manuellement.

La complexité à automatiser
Dans les cas où un système complexe doit être testé, l'automatisation peut présenter l'énorme avantage d'épargner au testeur la tâche difficile qui consiste à devoir répéter les étapes complexes qui sont fastidieuses, chronophages et sujettes à erreur d'exécution.

Cependant, certains scripts de test peuvent être difficiles ou non rentables à automatiser. De nombreux facteurs pourraient affecter ceci, y compris: un SUT qui n'est pas compatible avec les solutions existantes de tests automatisés ; l'obligation de produire une quantité de code conséquente et de développer des appels aux APIs dans le but d'automatiser ; la multiplicité des systèmes qui doivent être pris en compte dans le cadre de l'exécution d'un test ; l'interaction avec des interfaces externes et/ou des systèmes propriétaires ; les tests d'utilisabilité ; la quantité de temps nécessaire pour tester les scripts d'automatisation etc...

La compatibilité des outils en support
Il existe un large éventail de plates-formes de développement utilisées pour créer des applications. Le défi pour le testeur est de savoir quels outils de test disponibles existent (le cas échéant) pour une plate-forme donnée, et dans quelle mesure la plate-forme est supportée. Les organisations utilisent une variété d'outils de tests, commerciaux, open source, et d'autres développés en interne. Chaque organisation a des besoins et des ressources différentes pour utiliser les outils de test. Les fournisseurs prévoient généralement une assistance payante, et dans le cas des leaders du marché, ont généralement un écosystème d'experts qui peuvent aider à la mise en œuvre de l'outil de test. Des outils open source offrent un support limité tel que les forums en ligne dans lesquels les utilisateurs peuvent obtenir des informations et soumettre des questions. Pour les outils de test développés en interne, le personnel existant fournit le support.

La question de la compatibilité de l'outil de test ne doit pas être sous-estimée. S'embarquer sur un projet d'automatisation de test sans tout à fait comprendre le niveau de compatibilité entre les outils de test et le SUT peut avoir des résultats désastreux. Même si la plupart des tests sur le SUT peuvent être automatisés, il pourrait y avoir une situation où les tests les plus critiques ne le peuvent pas.

Maturité des processus de test
Afin de mettre en œuvre efficacement l'automatisation dans un processus de test , le processus en question doit être structuré, discipliné et répétable. L'automatisation entraine un processus de développement complet au sein du processus de test existant qui requiert de gérer le code d'automatisation et les composants connexes.

Adéquation de l'automatisation avec l'étape du cycle de vie du logiciel
Un SUT a un cycle de vie qui peut s'étendre sur des années, voire des décennies. Quand le développement d'un système commence, le système se transforme et se développe pour résoudre les défauts et ajouter des améliorations qui répondent aux besoins de l'utilisateur final.
Dans les premiers stades du développement d'un système, les changements peuvent être trop rapides pour mettre en œuvre une solution de tests automatisés. Quand les mises en page et les contrôles affichés à l'écran sont optimisés et améliorés, automatiser dans un environnement changeant et

dynamique peut exiger un travail continu, ce qui n'est pas efficace ni pertinent. Ce serait semblable à essayer de changer un pneu sur une voiture en mouvement ; il est préférable d'attendre que la voiture s'arrête. Pour les grands systèmes dans un environnement de développement séquentiel, lorsqu'un système s'est stabilisé et a un noyau de fonctionnalités, c'est alors un meilleur moment pour commencer la mise en œuvre de tests automatisés.

Au fil du temps, les systèmes atteignent la fin de leur cycle de vie et sont soit retirés soit reconçus afin d'utiliser des technologies plus récentes et plus efficaces. L'automatisation n'est pas recommandée pour un système approchant la fin de son cycle de vie car cette initiative à court terme n'apportera que peu de valeur à l'entreprise. Cependant, pour les systèmes reconçus avec une architecture différente mais qui gardent leurs fonctionnalités, un environnement de test automatisé qui contient des données sera utile tout autant pour les anciens systèmes que pour les nouveaux. Dans ce cas, la réutilisation de données de test sera possible et le recodage de l'environnement automatisé sera nécessaire pour être compatible avec la nouvelle architecture.

La durabilité de l'environnement automatisé
Un environnement de test automatisé doit être flexible et s'adapter aux changements du SUT qui interviendront au fil du temps. Cela comprend la capacité de diagnostiquer et de corriger les problèmes d'automatisation rapidement, la facilité avec laquelle les composants d'automatisation peuvent être maintenus et la facilité avec laquelle de nouvelles fonctionnalités et du support peuvent être ajoutés dans l'environnement automatisé. Ces éléments font partie intégrante de la conception globale et de la mise en œuvre de la gTAA.

La contrôlabilité du SUT (préconditions, configuration et stabilité)
Le TAE devrait identifier les caractéristiques de contrôle et de visibilité dans le SUT qui aideront à la création de tests automatisés efficaces. Sinon, l'automatisation des tests se base uniquement sur les interactions avec l'interface utilisateur, ce qui résulte dans une solution d'automatisation de test non maintenable. Voir la section 2.3 sur Conception pour testabilité et automatisation pour plus d'informations

Planification technique en support à l'analyse de ROI
L'automatisation de test peut apporter différents degrés de bénéfices à une équipe de test. Toutefois, un niveau significatif d'effort et de cout est associé à la mise en œuvre d'une solution de test automatisé efficace. Avant d'engager le temps et les efforts nécessaires pour développer des tests automatisés, une évaluation doit être menée afin d'évaluer globalement les bénéfices et les résultats potentiels et attendus de la mise en œuvre de l'automatisation de test.
Une fois cela déterminé, il faut définir les activités nécessaires pour réaliser un tel plan et les couts associés doivent être déterminés afin de calculer le ROI (Retour sur Investissement).

De façon à préparer comme il se doit la transition vers l'automatisation, les éléments suivants doivent être traités.

- Disponibilité d'outils d'automatisation dans l'environnement de test
- Données de test et cas de test corrects
- Périmètre de l'automatisation
- Connaissance de l'équipe de test sur le changement de concept
- Rôles et responsabilités
- Coopération entre développeurs et ingénieurs d'automatisation des tests
- Effort parallèle
- Reporting de l'automatisation

Disponibilité des outils dans l'environnement de test pour l'automatisation des tests

Les outils de test sélectionnés doivent être installés et leur fonctionnement vérifié dans l'environnement de test. Cela peut impliquer le téléchargement de tous services packs ou mises à jour, de sélectionner la configuration d'installation appropriée—incluant les add-ins—nécessaires pour supporter la SUT, et s'assurer que la TAS fonctionne correctement dans l'environnement par rapport à l'environnement de développement de l'automatisation.

Exactitude des données de test et des cas de test
L'exactitude et la complétude des données de test et des cas de test manuels est nécessaire pour s'assurer que leur utilisation avec l'automatisation fournira des résultats prédictibles. Les tests exécutés automatiquement ont besoin de données explicites en entrée, navigation, synchronisation, et validation.

Périmètre de l'effort d'automatisation
De façon à afficher des succès au plus tôt dans l'automatisation et obtenir des retours sur des problèmes techniques qui peuvent impacter l'avancement, commencer avec un périmètre réduit facilitera les tâches d'automatisation futures. Un projet pilote peut cibler une partie des fonctionnalités du système qui sont représentatives de l'interopérabilité du système total. Les leçons apprises du pilote vont aider à ajuster les estimations de durées et les calendriers futurs, et à identifier les parties nécessitant des ressources techniques spécialisées. Un projet pilote est un moyen facile de montrer le succès de l'automatisation au plus tôt, ce qui permettra d'avoir l'appui du management dans le futur.

Pour y arriver, les cas de test à automatiser doivent être sélectionnés de façon avisée. Sélectionner les cas qui requièrent peu d'effort d'automatisation, mais qui fournissent une grande valeur ajoutée. Les tests de régression ou les « smoke tests » peuvent être implémentés et apporter une valeur considérable car ces tests sont exécutés plutôt souvent, voire quotidiennement. Les tests de fiabilité sont d'autres bons candidats pour commencer. Ces tests sont souvent composés d'étapes relativement faciles et sont exécutés et ré exécutés, révélant des problèmes difficiles à détecter manuellement. Ces tests de fiabilité ne consomment pas trop d'effort de mise en œuvre, mais peuvent montrer une valeur ajoutée très tôt.

Ces projets pilotes mettent l'automatisation en lumière (Économie d'effort de test manuel, ou identification de problèmes sérieux) et pavent le chemin pour des extensions futures (effort et argent).

De plus, la priorisation devrait être donnée aux tests critiques pour l'organisation car ils apporteront la plus grande valeur ajoutée au début. Cependant, dans ce contexte, il est important que les tests les plus difficiles techniquement à automatiser soient évités. Sinon, trop d'effort sera dépensé à essayer de développer l'automatisation avec trop peu de résultats à afficher. En règle générale, identifier les tests sur les parties aux caractéristiques les plus communes de l'application donnera l'élan nécessaire pour continuer l'effort d'automatisation.

Connaissance de l'équipe de test dans le changement de concept
Les testeurs viennent de différents horizons: Certains sont des experts Métier de la communauté des utilisateurs finaux, ou analystes Métier, alors que d'autres ont de grandes connaissances techniques qui leur permettent de mieux comprendre l'architecture du système. Pour que le test soit efficace, un large pannel de profils est préférable. Lorsque l'équipe passe à l'automatisation, les rôles deviennent plus spécialisés. Changer la structure de l'équipe est essentiel pour que l'automatisation soit réussie, et former l'équipe au plus tôt va permettre de réduire l'anxiété concernant les rôles ou les craintes d'être redondants. Si cela est traité correctement, le passage à l'automatisation devrait rendre chaque membre de l'équipe de test très excité et prêt à participer au changement organisationnel et technique.

Rôles et responsabilités

L'automatisation de test devrait être une activité à laquelle tout le monde peut participer. Cependant, cela ne signifie pas que tout le monde a le même rôle. Concevoir, implémenter, et maintenir un environnement de test automatisé est technique par nature, et comme tel devrait être réservé aux personnes avec de fortes compétences en programmation et un grand vécu technique. Le développement de test automatisé devrait résulter dans un environnement utilisable aussi bien par des individus techniques que non techniques. De façon à maximiser la valeur d'un environnement de test automatisé il y a besoin de personnes qui connaissent le domaine et les tests car il sera nécessaire de développer les scripts de test appropriés (incluant les données de test associées). Ils seront utilisés pour piloter l'environnement automatisé et fournir la couverture de test prévue. Les experts métier passent en revue les rapports pour confirmer les fonctionnalités de l'application, alors que les experts techniques s'assurent que l'environnement automatisé fonctionne correctement et efficacement. Ces experts techniques peuvent également être des développeurs intéressés dans les tests. Les développeurs sont bons pour concevoir des logiciels maintenables, et cela est de la plus grande importance pour l'automatisation de test. Les développeurs peuvent se focaliser sur le Framework d'automatisation ou les bibliothèques de test. Les testeurs devraient conserver l'implémentation des cas de test.

Coopération entre développeurs et ingénieurs en automatisation des tests
Une automatisation des tests réussie requiert également l'implication de l'équipe de développement logicielle tout autant que des testeurs. Dans le passé, développeurs et testeurs avaient un minimum de contacts pendant le développement des tests. Les développeurs et les testeurs devront travailler plus étroitement pour l'automatisation de test pour que les développeurs fournissent le support et les informations techniques sur leurs méthodes de développements et outils. Les ingénieurs d'automatisation de test peuvent être concernés par la testabilité du code du développeur. Cela sera spécialement le cas si aucune norme n'est suivie, ou si les développeurs utilisent des bibliothèques/objets très récents ou maison ou particuliers. Par exemple, les développeurs peuvent choisir un contrôle «grille» d'une tierce partie qui peut ne pas être compatible avec l'outil d'automatisation sélectionné. Finalement, une équipe de gestion de projets doit avoir une compréhension claire des types de rôles et responsabilités requis pour réussir l'automatisation.

Effort parallèle
Comme parties des activités de transition, de nombreuses organisations créent une équipe parallèle pour commencer à automatiser les scripts de test manuels existants. Les nouveaux scripts automatisés sont alors intégrés dans les tests et remplacent les tests manuels. Cependant, avant que ce ne soit fait, il est souvent recommandé de comparer et de valider que les scripts automatisés réalisent les mêmes tests et apporte la même validation que le script manuel qui est remplacé.

La plupart du temps, une évaluation des scripts manuels sera faite avant la conversion vers l'automatisation. Suite à cette évaluation, il peut être déterminé qu'il y a besoin de restructurer les scripts manuels existants avec une approche plus efficace et efficiente.

Reporting sur l'automatisation
Différents rapports peuvent être générés automatiquement par une TAS. Cela inclus les statuts passé/échec des scripts individuels ou des étapes d'un script, les statistiques générales d'exécution, et les performances globales de la TAS. Il est également important d'avoir de la visibilité sur la qualité du fonctionnement de la TAS de façon à ce que chaque résultat spécifique de l'application qui est reporté puisse être réputé précis et complet (Voir Chapitre 7: Vérifier la TAS).

6.2 Identifier les étapes nécessaires pour implémenter l'automatisation des tests de régression

Les tests de régression sont opportuns pour l'automatisation. Un test fonctionnel d'aujourd'hui deviendra demain un test de régression. Ce n'est qu'une question de temps pour que l'automatisation de tests de régression devienne plus bénéfique que le temps passé par les ressources d'une équipe de test manuel traditionnelle.

Pendant les étapes de développement pour la préparation des tests de régression certaines questions doivent être posées:

- A quelle fréquence les tests sont-ils exécutés?
- Quelle est la durée d'exécution de chaque test, de chaque suite de non régression?
- Est ce qu'il y a chevauchement entre les tests?
- Est-ce que les tests partagent des données?
- Est que les tests sont dépendants les uns des autres?
- Quelles sont les préconditions requises avant l'exécution des tests?
- Quel pourcentage de la couverture du SUT les tests représentent-ils?
- Est-ce que les tests s'exécutent actuellement sans problème?
- Qu'est ce qui devrait être fait si les tests de régression prennent trop de temps?

Ces points sont traités plus en détail ci-dessous.

Fréquence d'exécution des tests
Les tests qui sont souvent exécutés en test de régression sont les meilleurs candidats à l'automatisation. Ces tests ont déjà été développés, exécutés sur les fonctionnalités connues du SUT, et verront leur temps d'exécution fortement réduit par l'automatisation.

Durée d'exécution des Tests
Le temps pris pour exécuter un test donné ou une suite de test complète est un paramètre important pour évaluer la valeur de l'implémentation de tests automatisés pour la régression. Commencer en automatisant les tests chronophages. Cela permettra à chaque test d'être exécuté plus rapidement et efficacement, et permettra d'ajouter des cycles d'exécution de tests automatisés. A ces bénéfices s'ajoutent des feedbacks plus fréquents sur la qualité du SUT et une réduction des risques de déploiement.

Recouvrement fonctionnel
Lors de l'automatisation des tests de régression, une bonne pratique est d'identifier les recouvrements fonctionnels qui existent entre les cas de test et, quand c'est possible, de réduire ces recouvrements dans leur équivalent automatisé. Cela permettra une meilleure efficience du temps d'exécution de ces tests automatisés, et se verra de plus en plus en avec l'accroissement du nombre de tests automatisés exécutés. Souvent, les tests développés en utilisant l'automatisation auront une nouvelle structure puisqu'ils dépendent de composants réutilisables et de banques de données partagées. Il n'est pas inhabituel de décomposer des tests manuels existants en différents tests plus petits. Tout comme, la consolidation de différents tests manuels en des tests automatisés plus gros peut être la solution appropriée. Les tests manuels doivent être évalués individuellement, et dans leur ensemble, de façon à ce qu'une stratégie de conversion efficace puisse être développée.

Partage des données

Les tests partagent souvent les mêmes données. Cela peut arriver quand les tests utilisent le même ensemble de données pour tester différentes fonctionnalités du SUT. Un exemple peut être un cas de tests A qui vérifie les vacances disponibles d'un employé, quand le cas de test B vérifie quelles formations cet employé a suivi dans le cadre de son développement de carrière. Chaque cas de test utilise le même employé, mais vérifie des paramètres différents. Dans un environnement de test manuel, les données de l'employé devraient être saisies plusieurs fois au cours de chaque cas de test manuel concernant cet employé. Alors que, dans un test automatisé, les données qui sont partagées devraient —quand c'est possible et faisable — être stockées et accessibles dans une source unique pour éviter la duplication, ou l'introduction d'erreurs.

Interdépendance des tests
Quand des scénarios de test de régression complexes sont exécutés, un test peut avoir des dépendances avec un ou plusieurs autres tests. Cela peut être habituel, et peut arriver par exemple, quand un nouveau numéro de commande est créé par une étape de test. Les tests suivant peuvent vouloir vérifier que : a) la nouvelle commande est affichée correctement par le système, b) Les modifications de la commande sont possibles, ou c) qu'effacer la commande est possible. Dans chaque cas, le numéro de commande qui a été créé dynamiquement au cours du premier test, doit être capturé pour être réutilisé par des tests ultérieurs. En fonction de la conception de la TAS, cela peut être traité.

Conditions préalables aux tests
Souvent, un test ne peut pas être exécuté avant le paramétrage des conditions initiales. Ces conditions peuvent inclure la sélection de la bonne base de données ou les jeux de données de test avec lesquels tester, ou le paramétrage initial de valeurs ou de paramètres. Beaucoup de ces étapes d'initialisation requises pour établir les préconditions peuvent être automatisées. Ne pas oublier de faire ces étapes avant l'exécution des tests permet d'obtenir une solution plus fiable et sûre. Comme ces tests de régression sont automatisés, ces préconditions doivent faire partie du processus d'automatisation.

Couverture du SUT
A chaque fois que des tests sont exécutés, une partie de la fonctionnalité d'un SUT est testée. Afin de vérifier la qualité globale du SUT, les tests doivent être conçus de manière à avoir la couverture la plus large et la plus profonde possible. En outre, des outils de couverture de code peuvent être utilisés pour surveiller l'exécution des tests automatisés et aider à quantifier l'efficacité des tests. Au fil du temps, grâce à des tests de régression automatisés, nous pouvons considérer que des tests supplémentaires fournissent une couverture supplémentaire. Cette mesure fournit un moyen efficace pour la quantification de la valeur des tests eux-mêmes.

Tests exécutables
Avant de convertir un test manuel de régression en un test automatisé, il est important de vérifier que le test manuel fonctionne correctement. Cela donne alors le bon point de départ pour assurer une conversion réussie d'un test de régression automatisé. Si le test manuel ne s'effectue pas correctement - soit parce qu'il a été mal écrit, soit parce qu'il utilise des données invalides, soit parce qu'il est obsolète ou hors de synchronisation avec le SUT en cours, ou en raison d'un défaut de SUT - le convertir à l'automatisation avant la compréhension et/ou la résolution de la cause de l'échec va créer un non-fonctionnement du test automatisé, ce qui est inutile et non-productif.

Les grands ensembles de test de régression
L'ensemble des tests de régression d'un SUT peut devenir très grand, si grand que l'ensemble des tests ne puisse pas être complètement exécuté la nuit ou le week-end. Dans ce cas, l'exécution simultanée de cas de tests est une possibilité si plusieurs SUT sont disponibles (pour les applications PC, cela ne pose probablement pas de problème, mais quand le SUT consiste en un avion ou une fusée c'est une autre

histoire !). Les SUT peuvent être rares et/ou couteux ce qui fait de l'exécution en concurrence une option irréaliste. Dans ce cas, il peut être possible d'exécuter seulement certaines parties du test de régression. Le choix de la partie de la suite de test de régression à exécuter peut également être basé sur une analyse des risques (Quelles parties du SUT ont été changées dernièrement ?).

6.3 Facteurs à considérer lors de l'automatisation de tests de nouvelles fonctionnalités

En général, il est plus facile d'automatiser des cas de test pour les nouvelles fonctionnalités alors que la mise en œuvre n'est pas encore terminée (ou mieux : pas encore commencée). L'ingénieur de test peut utiliser ses connaissances pour expliquer exactement aux développeurs et architectes ce qui est nécessaire dans la nouvelle fonctionnalité pour qu'elle puisse être testée efficacement par la TAS.

Comme de nouvelles fonctionnalités sont introduites dans le SUT, les testeurs doivent développer des nouveaux tests pour ces fonctionnalités et exigences correspondantes. Le TAE doit solliciter les commentaires des concepteurs de test suivant leur domaine d'expertise et déterminer si la TAS actuelle répond aux besoins des nouvelles fonctionnalités. Cette analyse inclut, mais n'est pas limitée à, l'approche actuelle utilisée, les outils de développement tiers, les outils de test utilisés, etc...

Les modifications apportées à la TAS doivent être évaluées par rapport aux composants testware automatisés existants afin que les modifications ou ajouts soit entièrement documentés, et n' affectent pas le comportement (ou la performance) de la fonctionnalité existante de la TAS.

Si une nouvelle fonctionnalité est implémentée avec, par exemple, une autre classe d'un objet, il peut être nécessaire de faire des mises à jour ou des ajouts aux composants du testware. En outre, la compatibilité avec les outils de test existants doit être évaluée et, le cas échéant, des solutions alternatives doivent être identifiées. Par exemple, si une approche pilotée par mots-clés est utilisée, développer des mots-clés supplémentaires ou modifier / élargir les existants pour accueillir la nouvelle fonctionnalité peut être nécessaire. Il peut être nécessaire d'évaluer des outils de test supplémentaires pour prendre en charge le nouvel environnement sous lequel la nouvelle fonctionnalité existe. Par exemple, un nouvel outil de test peut être nécessaire si l'outil de test existant ne prend en charge que le code HTML.

Les nouvelles exigences de test peuvent affecter les tests automatisés et des composants de testware existants. Par conséquent, avant de faire des changements, les tests automatisés existants doivent être exécutés sur le nouveau SUT / mise à jour afin de vérifier et d'enregistrer tout changement nécessaire au bon fonctionnement des tests automatisés existants. Cela devrait inclure la cartographie des interdépendances à d'autres tests. Les nouveaux changements technologiques nécessiteront d'évaluer les composants de testware actuels (y compris les outils de test, les bibliothèques de fonctions, API, etc.) et la compatibilité avec les TAS existants.

Lorsque les exigences actuelles changent, l'effort de mise à jour des cas de tests qui vérifient ces exigences devrait faire partie de l'échéancier du projet (structure de répartition du travail). La traçabilité entre exigences et cas de test indiquera quels cas de tests doivent être mis à jour.

Enfin, il faut déterminer si la TAS actuelle continuera à répondre aux besoins du SUT actuel. Les techniques de mise en œuvre sont-elles toujours valables, ou est-ce qu'une nouvelle architecture est nécessaire, cela ne peut-il pas être fait en étendant la capacité actuelle ?

Lorsque de nouvelles fonctionnalités sont introduites, c'est une occasion pour les ingénieurs de test de s'assurer que la fonctionnalité nouvellement définie sera testable. Au cours de la phase de conception, les tests devraient être pris en compte en prévoyant de fournir des interfaces de test qui peuvent être utilisées par les langages de script ou par l'outil d'automatisation de test et ce afin de vérifier les nouvelles fonctionnalités. Pour plus d'informations, Voir Section 2.3, Conception pour testabilité et automatisation.

6.4 Facteurs à considérer lors de l'automatisation des tests de confirmation

Un test de confirmation se produit lorsqu'un correctif de code qui traite une anomalie signalée a été fourni. Un testeur suit généralement les étapes nécessaires pour reproduire le défaut et vérifier que le défaut n' existe plus.

Les défauts peuvent trouver un moyen de se réintroduire dans les versions ultérieures (cela peut indiquer un problème de gestion de la configuration) et, par conséquent, les tests de confirmation sont les premiers candidats à l'automatisation. L'utilisation de l'automatisation permet de réduire le temps d'exécution des tests de confirmation. Le test de confirmation peut être ajouté et compléter le banc d'essai de régression automatisé existant.
Le test automatisé des défauts a généralement une portée limitée de fonctionnalités. Au fil du temps, cette fonctionnalité peut demeurer comme un test indépendant ou être intégrée dans un test automatisé existant qui couvre des fonctionnalités similaires ou plus larges. Avec cette approche, la valeur de l'automatisation des tests de confirmation tient toujours.

Le suivi des tests de confirmation automatisés permet d'obtenir des rapports supplémentaires sur le temps et le nombre de cycles dépensés pour résoudre les défauts.

En plus du test de confirmation, le test de régression est nécessaire pour s'assurer que de nouveaux défauts n'ont pas été introduits comme un effet secondaire de la correction de défauts. Une analyse d'impact peut être nécessaire pour déterminer la portée appropriée des tests de régression.

7 Vérification de la TAS - 120 mins.

Mots clés
vérification

Objectifs d'apprentissage pour vérification de la TAS

7.1 Vérification des composants de l'environnement de test automatisé

ALTA-E-7.1.1 (K3) Appliquer les contrôles de validité à l'environnement de test automatisé et à la configuration de l'outil de test

7.2 Vérification de la suite de tests automatisés

ALTA-E-7.2.1 (K3) Vérifier le comportement correct d'un script de test automatisé et/ou d'une suite de test donnée

7.1 Vérification des composants de l'environnement de test automatisé

L'équipe d'automatisation de test doit être en mesure de vérifier que l'environnement de test automatisé fonctionne comme prévu. Ces vérifications sont effectuées, par exemple, avant de commencer les tests automatisés ou lorsque l'on vérifie que l'environnement fonctionne toujours correctement.

Il existe un certain nombre d'étapes afin de vérifier les composants de l'environnement de tests automatisés. Chacune de ces étapes est expliquée plus en détail ci-après.

Installation, initialisation, configuration et customisation de l'outil de test
La TAS comporte de nombreux composants. Chacun de ces composants a besoin d'être identifié pour assurer une performance fiable et reproductible. Au cœur d'une tâche se trouvent les composants exécutables, correspondant aux fichiers de données, aux bibliothèques fonctionnelles, et aux fichiers de données et de configuration.
Le processus de configuration d'une TAS peut aller de l'utilisation de scripts d'installation automatisés au placement manuel de dossiers dans des dossiers correspondants. Les outils de test, tout comme les systèmes d'exploitation et d'autres applications, ont régulièrement des service packs ou peuvent avoir des compléments optionnels ou requis pour s'assurer de la compatibilité avec n'importe quel environnement d'un SUT donné.

Une installation (ou copie) automatisée à partir d'un répertoire central a des avantages. Cela permet de s'assurer que les tests de plusieurs SUT ont été exécutés avec la même version et la même configuration de la TAS. Les mises à jour de la TAS peuvent être effectuées avec le répertoire. L'usage du répertoire et le processus pour améliorer une nouvelle version de la TAS doivent être les mêmes que pour des outils de développement standards.

L' exécution de scripts de "tests" avec les succès et les échecs connus
Lorsque des cas de test connus pour fonctionner échouent, il est clair que quelque chose est fondamentalement erroné et doit être corrigé dès que possible. Inversement, lorsque des cas de test passent même alors qu'ils auraient dû échouer, nous devons identifier le composant qui n'a pas fonctionné correctement. Il est important de vérifier la génération correcte des fichiers d'enregistrement, les données de performance, l'installation et le démontage du cas de test / script. Il est également utile d'exécuter quelques tests des différents types de test (tests fonctionnels, tests de performance, tests de composants etc.). Cela devrait également être effectué au niveau du framework.

Répétabilité dans la mise en place/destruction de l'environnement de test
Une TAS sera implémentée dans une variété de systèmes et de serveurs. Pour s'assurer que la TAS fonctionne correctement dans chaque environnement, il faut avoir une approche systématique du chargement et du déchargement de la TAS dans tous les environnements donnés. Ceci est réussi quand la construction et la reconstruction de la TAS ne fournissent aucune différence perceptible dans la manière de fonctionner dans et au travers de multiples environnements. La gestion de la configuration des composants de la TAS permet de s'assurer qu'une configuration donnée peut être créée de façon fiable.

Configuration de l'environnement de test et des composants
Pour avoir les connaissances nécessaires sur les différents aspects d'une TAS qui peuvent être affectés ou nécessiter des changements quand l'environnement du SUT change, il faut comprendre et documenter les différents composants qui composent la TAS.

Connectivité sur les systèmes/interfaces internes et externes

Une fois qu'une TAS est installée dans l'environnement d'un SUT donné, et avant de l'utiliser sur le SUT, un ensemble de tests ou de pré-tests doivent être faits pour s'assurer que les connexions entre les systèmes internes et externes, les interfaces, etc. sont disponibles. Il est essentiel d'établir des conditions de test pré-automatisées pour s'assurer que la TAS a été installée et configurée correctement.

Intrusivité des outils de tests automatisés dans l'environnement de test
Souvent, la TAS sera étroitement liée au SUT. Ceci est intentionnel et c'est pourquoi il y a un haut niveau de compatibilité, surtout quand cela concerne les interactions au niveau de l'IHM. Toutefois, cette intégration étroite peut aussi avoir des effets négatifs. Par exemple: le SUT se comporte différemment quand la TAS est dans l'environnement du SUT, l'exécution de la TAS sur le SUT crée un comportement différent par rapport à une utilisation manuelle du SUT, la performance du SUT est affectée avec la TAS dans l'environnement, ou bien lorsqu'on exécute la TAS sur le SUT.

Le niveau d'intrusivité diffère selon l'approche de test automatisé choisie. Par exemple:

- Lorsque l'on communique avec le SUT depuis des interfaces externes, le niveau d'intrusion sera très bas. Les interfaces externes peuvent être des signaux électroniques (pour les interrupteurs physiques), des signaux USB pour des périphériques USB (comme les claviers). Grâce à cette approche, la simulation de l'utilisateur final est la meilleure. De plus, le logiciel du SUT n'est pas modifié pour des objectifs de test. Le comportement et le timing du SUT ne sont pas influencés par l'approche de test. Communiquer avec le SUT de cette manière peut être complexe. Du hardware spécifique peut être nécessaire, des langages de description du hardware nécessaires pour se connecter au SUT, etc. Pour les systèmes uniquement logiciels, cette approche n'est pas courante, mais elle l'est plus pour des produits d'informatique embarquée.
- Lorsque l'on communique avec le SUT au niveau de l'interface graphique, l'environnement du SUT est adapté pour que l'on puisse injecter des commandes ld'IHM et extraire les informations nécessaires pour les cas de test. Le comportement du SUT n'est pas directement modifié, mais son timing est affecté et cela peut avoir un impact sur le comportement. Le niveau d'intrusion est plus élevé que dans la situation précédente mais il est moins complexe de communiquer avec le SUT de cette manière. Souvent, des outils commerciaux sur étagères peuvent être utilisés pour ce type d'automatisation.
- La communication avec le SUT peut être effectuée via des interfaces de test dans le logiciel ou en utilisant des interfaces existantes qui sont déjà fournies par le logiciel. La disponibilité de ces interfaces (Interfaces de Programmation de l'Application) est importante dans la conception de la testabilité. Le niveau d'intrusion peut être assez élevé ici. Les tests automatisés utilisent des interfaces qui peuvent ne pas être utilisées du tout par les utilisateurs finaux du système (interfaces de test) ou les interfaces qui peuvent être utilisées dans un autre contexte et non dans le monde réel. D'un autre côté, il est très facile d'effectuer des tests automatisés via des interfaces de programmation(API), et de plus cela ne coute pas cher. Tester le SUT via des interfaces de test peut être une approche solide tant que les risques potentiels sont compris.

Un haut niveau d'intrusion peut générer des défaillances pendant le test, qui n'apparaîtraient pas dans des conditions d'utilisation réelles. Quand beaucoup de défaillances, qui n'apparaîtraient pas dans des conditions d'utilisation réelles, sont identifiées avec les tests automatisés, la confiance que l'on a dans la TAS peut baisser drastiquement. A la fin, les développeurs demanderont que les défaillances identifiées par le test automatisé soient d'abord reproduites manuellement avant que le problème ne soit analysé et résolu.

Tester les caractéristiques de la structure

Le framework peut fournir différentes caractéristiques générales qui peuvent être utilisées par les cas de test: vérifier les fuites de mémoire, vérifier les conditions d'erreurs inattendues manquées par les cas de test, la dégradation de la performance, les comparaisons complexes, etc. Ces caractéristiques doivent être explicitement testées car les cas de test disponibles et récemment développés dépendent de cette fonctionnalité de la structure.

7.2 Vérification de la Suite de tests automatisés

Les suites de tests automatisés doivent être testées au niveau de leur cohérence et de leur comportement. On peut appliquer différents types de vérification de validité pour s'assurer que la suite de tests automatisés est opérationnelle et exécutable à n'importe quel moment, ou pour déterminer qu'elle est prête à l'emploi.

Plusieurs étapes sont à suivre pour vérifier la suite de tests automatisés. Celles-ci incluent :

- Exécuter les scripts de «test» avec les réussites et les échecs connus.
- Vérifier la suite de test.
- Vérifier les nouveaux tests concentrés sur les nouvelles caractéristiques du framework.
- Prendre en compte la répétabilité des tests.
- Vérifier qu'il y a assez de points de vérification dans la suite de tests automatisés.

Chacune de ces étapes est expliquée plus en détail ci-dessous.

Exécuter les scripts de « test » avec les réussites et les échecs connus
Quand un cas de test qui est connu pour fonctionner échoue, il est clair que quelque chose est fondamentalement erroné et doit être réparé aussi vite que possible. Inversement, quand des cas de test réussissent alors qu'ils auraient dû échouer, il faut identifier le composant qui n'a pas bien fonctionné. Il est important de vérifier que la génération des enregistrement est correcte, ainsi que les données de performance, la mise en place et la destruction du cas/script de test. Il est aussi utile d'exécuter quelques tests de types différents (tests fonctionnels, tests de performance, tests des composants, etc.). Cela doit aussi être effectué au niveau du framework.

Vérifier la suite de test
Vérifier que la suite de test est complète (tous les cas de test ont les résultats attendus, les données de test sont présentes), que la version est la bonne et que le framework et le SUT sont compatibles.

Vérifier les nouveaux tests qui se concentrent sur les nouvelles caractéristiques de la structure
La première fois qu'une nouvelle caractéristique de la TAS est utilisée dans des cas de test, elle doit être vérifiée et supervisée attentivement pour s'assurer que la caractéristique fonctionne correctement.

Prendre en compte la répétabilité des tests
Quand on répète des tests, le résultat/verdict du test doit toujours être le même. Si des cas de test dans l'ensemble de tests ne donnent pas un résultat fiable, ces tests sont du gaspillage et doivent être réparés le plus vite possible. On passera du temps plusieurs fois sur ces tests pour analyser le problème. Les cas de test présents dans le jeu de test qui ne donnent pas un résultat fiable (par exemple, les conditions de run) pourraient être retirés de la suite de tests automatisés en cours et analysés séparément afin de trouver la cause principale du problème. Sinon, du temps sera passé à plusieurs reprises sur ces séries de tests pour analyser le problème.

Quand les cas de test présentent des défaillances intermittentes, celles-ci doivent être analysées. Le problème peut se trouver dans le cas de test lui-même ou dans le framework (ou bien il peut même y avoir un problème dans le SUT). Une analyse des enregistrements d'exécution(du cas de test, de la structure et

du SUT) peut permettre d'identifier la source du problème. Il peut aussi être nécessaire de déboguer. L'aide de l'analyste de test, du développeur de test et de l'expert du domaine pour trouver la cause du problème peut être nécessaire.

Vérifier qu'il y a assez de points de vérification dans la suite de tests automatisés
Il doit être possible de vérifier que la suite de tests automatisés a été exécutée et qu'elle a obtenu les résultats attendus. Des preuves pour garantir que la suite de tests et/ou les cas de tests ont été effectués comme prévu doivent être fournies. Ces preuves incluent l'enregistrement du début et de la fin de chaque cas de test, l'enregistrement du statut d'exécution du test pour chaque cas de test terminé, la vérification que les conditions préalables ont été atteintes, etc.

8 Amélioration continue - 150 mins.

Mots-clés
maintenance

Objectifs d’apprentissage pour l’amélioration continue

8.1 Options pour améliorer l’automatisation de test

ALTA-E-8.1.1 (K4) Analysez les aspects techniques d'une solution d'automatisation de test déployée et fournissez des recommandations pour son amélioration

8.2 Adaptation de l'automatisation des tests à l'environnement et aux changements du SUT

ALTA-E-8.2.1 (K4) Analyser le testware automatisé, y compris les composants de l’environnement de test, les outils et les bibliothèques de fonctions, de façon à comprendre où la consolidation et les mises à jour doivent être faites en fonction d’un ensemble donné de changements d’environnement de test ou de modifications du SUT

8.1 Options pour améliorer l'automatisation de test

En plus des tâches de maintenance permanentes qui sont nécessaires pour que la TAS reste synchronisée avec le SUT, plusieurs opportunités peuvent se présenter pour l'améliorer. Ces améliorations peuvent être faites pour obtenir toutes sortes d'avantages, comme une plus grande efficacité (en réduisant encore plus les interventions manuelles), une meilleure facilité d'utilisation, des compétences supplémentaires et un meilleur support des activités de test. Le choix de la manière d'améliorer la TAS sera influencé par les avantages qui ajoutent le plus de valeur à un projet.

On peut envisager d'améliorer des domaines spécifiques d'une TAS, comme par exemple le scripting, la vérification, l'architecture, le pré-traitement et le post-traitement, la documentation et les outils de support. Cela est détaillé ci-dessous.

Le scripting
Les approches de développement peuvent aller d'une simple approche structurée à des approches plus sophistiquées pilotées par mots-clés, en passant par des approches pilotées par les données. Il peut être judicieux d'améliorer l'approche du scripting de la TAS actuelle et de passer à des nouveaux tests automatisés. L'approche peut être adaptée pour tous les tests automatisés existants, ou au moins pour ceux qui impliquent le plus de travail de maintenance.

Plutôt que de changer complètement l'approche de développement, les améliorations de la TAS peuvent se concentrer sur l'implémentation des scripts. Par exemple:

- Évaluer le chevauchement du cas/de l'étape/de la procédure de test afin de consolider les tests automatisés.
  Les cas de test qui contiennent des séquences d'actions et des vérifications similaires ne doivent pas implémenter ces étapes plusieurs fois. Ces étapes doivent être ajoutées dans une bibliothèque pour pouvoir être réutilisées. Cette bibliothèque peut alors être utilisée par les différents cas de test. Cela augmente la maintenabilité du testware. Quand des étapes ou la vérification du test ne sont pas identiques mais semblables, un paramétrage peut être nécessaire.
  Remarque: c'est une approche classique dans le test piloté par les mots-clés**Erreur ! Signet non défini.**.
- Établir un processus de recouvrement d'erreur pour la TAS et le SUT.
  Quand une erreur se produit lors de l'exécution des cas de test, la TAS doit pouvoir être restaurée suite à cette condition d'erreur pour pouvoir continuer avec le cas de test suivant. Quand une erreur se produit dans le SUT, la TAS doit pouvoir effectuer les actions de récupération nécessaires sur le SUT (un redémarrage du SUT complet par exemple).
- Évaluer les mécanismes d'attente pour s'assurer que le meilleur est utilisé.
  Il y a trois mécanismes d'attente habituels:
  1. Des attentes codées en dur (attendre un certain nombre de millisecondes) peuvent être la cause de beaucoup de problèmes d'automatisation de test.
  2. Une attente dynamique par polling– vérifier s'il y a eu un changement d'état ou si une action s'est produite par exemple – est bien plus flexible et efficace:
     - Seul le temps nécessaire est attendu et il n'y a pas de perte de temps.
     - Quand pour une raison quelconque le processus prend plus de temps que prévu, le polling durera jusqu'à que la condition soit vraie. Ne pas oublier d'inclure un timeout ou alors le test pourra éternellement rester dans un état d'attente s'il y a un problème.
  3. Adhérer au mécanisme d'évènement du SUT est encore plus efficace. Cette option est bien plus fiable que les deux précédentes, mais le langage de scripting de test doit

permettre la souscription de l'évènement et le SUT doit offrir ces évènements à l'application de test.

- Traiter le testware comme un logiciel.
  Le développement et la maintenance du testware sont juste des formes de développement logiciel. En tant que tel, les meilleures pratiques de codage (utiliser des directives de codage, une analyse statique, des revues de code, etc.) doivent être appliquées. Il peut même être judicieux d'utiliser des développeurs logiciel (à la place des ingénieurs de test) pour développer certaines parties du testware (les bibliothèques par exemple).
- Évaluer les scripts existants en vue d'une révision/retrait
  Plusieurs scripts peuvent poser des problèmes (Défaillances de temps à autre, couts de maintenance élevés, etc.), et il peut s'avérer utile de reconcevoir ces scripts. D'autres scripts de test peuvent être retirés de la suite car ils n'apportent plus aucune valeur.

Exécution des tests
Lorsqu'une suite de test de régression automatisée n'est pas terminée du jour au lendemain (durant la nuit), cela ne devrait pas être une surprise. Lorsque le test prend trop de temps, il peut être nécessaire de tester simultanément sur différents systèmes, mais ce n'est pas toujours possible. Lorsque des systèmes couteux (cibles) sont utilisés pour les tests, le fait que tous les tests doivent être effectués sur une seule cible peut constituer une contrainte. Il peut être nécessaire de diviser la suite de test de régression en plusieurs parties, chacune s'exécutant dans une période de temps définie (par exemple, en une seule nuit). Une analyse plus approfondie de la couverture des tests automatisés peut révéler des doublons. La suppression des doublons peut réduire le temps d'exécution et se révéler plus efficace.

Vérification
Avant de créer de nouvelles fonctions de vérification, il convient d'adopter une série de méthodes de vérifications standards qui seront utilisées par tous les tests automatisés. Cela permettra d'éviter la ré-implémentation d'actions de vérification sur de nombreux tests. Lorsque les méthodes de vérification ne sont pas identiques mais similaires, l'utilisation du paramétrage permet d'autoriser une fonction à utiliser sur plusieurs types d'objets.

Architecture
Il peut être nécessaire d'améliorer l'architecture pour améliorer la testabilité du SUT. Ces changements peuvent être effectués dans l'architecture du SUT et/ou dans l'architecture de l'automatisation. Ceci peut conduire à une amélioration importante de l'automatisation de test, mais peut nécessiter des changements significatifs (investissement) dans le SUT/la TAS. Par exemple, si le SUT doit être modifié pour fournir des API de test, la TAA doit pouvoir accéder à ces API. Cela peut être assez couteux d'ajouter tardivement ce genre de caractéristiques dans le processus, il est conseillé d'y penser au début de l'automatisation (et au début du développement du SUT – voir partie 2.3 Conception pour Testabilité et Automatisation).

Prétraitement et post-traitement
Fournir des tâches standard d'installation et de suppression. Celles-ci sont aussi appelées pré-traitement (installation) et post-traitement (suppression). Cela permet de ne pas répéter les tâches à plusieurs reprises pour chaque test automatisé, et de non seulement réduire les couts de maintenance mais également le travail requis pour implémenter les nouveaux tests automatisés.

Documentation

Cela couvre toutes les formes de documentation, de celle des scripts (ce qu'ils font, comment ils doivent être utilisés, etc.), en passant par la documentation utilisateur de la TAS, jusqu'aux rapports et enregistrements produits par la TAS.

Fonctionnalités de la TAS
Ajouter des caractéristiques et fonctions supplémentaires à la TAS comme un reporting détaillé, des enregistrements, l'intégration avec d'autres systèmes, etc. N'ajouter ces nouvelles caractéristiques que lorsque celles-ci seront utilisées par les cas de test. Ajouter des caractéristiques inutilisées ne fait qu'augmenter la complexité et diminue la fiabilité et la maintenabilité.

Mises à jour/améliorations de la TAS
En améliorant /mettant à jour la TAS, des nouvelles fonctions peuvent être disponibles et peuvent être utilisées par les cas de test (ou les défaillances peuvent être corrigées). Toutefois, la mise à jour de la structure (en améliorant les outils de test existants ou en introduisant des nouveaux) peut avoir une conséquence négative sur les cas de test existants. Il faut tester la nouvelle version de l'outil de test par des échantillons de test avant de lancer la nouvelle version. Les échantillons de test doivent être représentatifs des tests automatisés des différentes applications, types de test et, le cas échéant, des différents environnements.

8.2 Planification de la mise en œuvre des améliorations de l'automatisation de test

Modifier une TAS existante nécessite une planification et une vérification rigoureuses. Beaucoup d'efforts ont été investis dans la création d'une TAS robuste composée d'un TAF et de bibliothèques de composants. Tous les changements, même triviaux, peuvent avoir des répercutions de grande envergure sur la fiabilité et la performance de la TAS.

Identifier les changements dans les composants de l'environnement de test
Évaluer quels changements et améliorations doivent être faits. Nécessitent-ils de modifier le logiciel de test, de personnaliser les bibliothèques de fonctions, les systèmes d'exploitation ? Ils ont tous un impact sur la performance de la TAS. L'objectif principal est de s'assurer que les tests automatisés continuent à fonctionner efficacement. Les changements doivent être faits progressivement pour que l'impact sur la TAS puisse être mesuré grâce à une exécution limitée de scripts de test. Une fois qu'aucun défaut nuisible n'a été trouvé, les changements peuvent être complètement implémentés. Une exécution complète de tests de régression est la dernière étape avant la confirmation que les changements n'ont pas affecté de façon nuisible les scripts automatisés. Pendant l'exécution de ces scripts de régression, des erreurs peuvent être trouvées. En identifiant la cause de ces problèmes (avec un reporting, les journaux, une analyse de données, etc.), on pourra s'assurer qu'elles ne sont pas une conséquence de l'activité d'amélioration de l'automatisation.

Augmenter l'efficacité et la productivité des bibliothèques de fonctions de la TAS centrale
Lorsqu'une TAS mûrit, de nouveaux moyens pour effectuer les tâches plus efficacement sont découverts. Ces nouvelles techniques (qui incluent l'optimisation du code des fonctions, l'utilisation de DLL plus récentes, etc.) doivent être intégrées au cœur des bibliothèques de fonctions qui sont utilisées par le projet actuel et par tous les projets.

Cibler les diverses fonctions qui agissent sur le même type de contrôle pour la consolidation
Interroger les contrôles dans l'Interface Graphique est ce qui se produit le plus pendant l'exécution d'un test automatisé. Cette interrogation donne des informations sur ce contrôle (visible/non visible, activé/non activé, taille et dimensions, données, etc.). Avec ces informations, un test automatisé peut sélectionner un

élément à partir d'une liste déroulante, entrer des données dans un champ, lire une valeur à partir d'un champ, etc. Plusieurs fonctions peuvent agir sur les contrôles pour obtenir ces informations. Certaines fonctions sont très spécialisées alors que d'autres sont plus générales. Par exemple, une fonction spécifique qui fonctionne seulement sur les listes déroulantes, une autre fonction (ou elle peut être créée et utilisée dans la TAS) qui fonctionne avec plusieurs fonctions en spécifiant une fonction comme l'un de ses paramètres. Un Ingénieur d'Automatisation de Tests peut alors utiliser plusieurs fonctions qui peuvent être consolidées en moins de fonctions, en obtenant les mêmes résultats et en minimisant les exigences de maintenance.

Refactoriser la TAA pour prendre en compte les évolutions du SUT
Pendant la durée de vie d'une TAS, des changements devront être effectués pour prendre en compte les évolutions du SUT. Quand le SUT évolue et mûrit, la TAA associée devra aussi évoluer si elle veut être capable de supporter le SUT. Il faut faire attention lorsqu'on étend les caractéristiques pour qu'elles ne soient pas implémentées d'une manière figée mais qu'elles soient analysées et changées au niveau de l'architecture de la solution automatisée. Cela permettra de garantir que quand une nouvelle fonctionnalité du SUT est requise, les composants compatibles seront en place pour tenir compte de ces nouveaux tests automatisés.

Conventions de nommage et standardisation
Comme des changements sont introduits, les conventions de nommage pour le nouveau code d'automatisation et les nouvelles bibliothèques de fonctions doivent être cohérentes avec les standards définis précédemment (voir partie 4.4.2 Cadre et Approche).

Évaluation des scripts existants pour la révision/retrait du SUT
Le processus de changement et d'amélioration inclut également une évaluation des scripts existants, leurs utilisations et leurs valeurs continues. Par exemple, si certains tests sont complexes et chronophages à exécuter, les décomposer en plusieurs tests plus petits peut être viable et efficace. Cibler les tests qui ne fonctionnent que rarement ou pas du tout pour pouvoir les éliminer et diminuer ainsi la complexité de la TAS et apporter une meilleure visibilité de ce qu'il est nécessaire de maintenir.

9 Références

9.1 Normes

Les Normes pour l'automatisation de test incluent mais ce n'est pas limités à:

- The Testing and Test Control Notation (TTCN-3) by ETSI (European Telecommunication Normes 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 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
- Le standard ISO/IEC/IEEE 29119-3
- Le "UML Testing Profile" (UTP) par OMG (Object Management Group) spécifiant les concepts de spécification des tests pour
  - L'architecture de test
  - Les données de Test
  - Le comportement des Tests
  - L'enregistrement des Tests
  - La gestion des Tests

9.2 Documents ISTQB

Identifiant	Référence

ISTQB-AL-TM	Tester Certifié CFTL/ISTQB, Syllabus Niveau Avancé, Test Manager, Version 2012, disponible sur [CFTL-Web]
ISTQB-AL-TTA	Tester Certifié CFTL/ISTQB, Syllabus Niveau Avancé, Analyst Technique de Test, Version 2012, disponible sur [CFTL-Web]
ISTQB-EL-CEP	Extension de la Certification du Niveau Expert CFTL/ISTQB, disponible sur [CFTL-Web]
ISTQB-EL-Modules	Vue Générale des modules CFTL/ISTQB Niveau Expert, Version 1.2, 23 Août 2013, disponible sur [CFTL-Web]
ISTQB-EL-TM	Niveau Expert CFTL/ISTQB – Syllabus Management des Tests, Version 2011, disponible sur [CFTL-Web]
ISTQB-FL	Syllabus CFTL/ISTQB Niveau Fondation, Version 2011, disponible sur [CFTL-Web]
ISTQB-Glossary	Glossaire CFTL/ISTQB des termes des tests logiciels, Version 2.4, 4 juillet 2014, disponible sur [CFTL-Web]

9.3 Marques

Les marques commerciales et marques de service déposées suivantes sont utilisées dans ce document:

ISTQB® est une marque déposée de l'the International Software Testing Qualifications Board

9.4 Livres

Identifiant	Référence de l'ouvrage
[Baker08]	Paul Baker, Zhen Ru Dai, Jens Grabowski et Ina Schieferdecker, "Model-Driven Testing: Using the UML Testing Profile", Edition Springer 2008, ISBN-10: 3540725628, ISBN-13: 978-3540725626
[Dustin09]	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
[Dustin99]	Efriede Dustin, Jeff Rashka, John Paul, "Automated Software Testing: introduction, management, and performance", Addison-Wesley, 1999, ISBN-10: 0201432870, ISBN-13: 9780201432879
[Fewster&Graham12]	Mark Fewster, Dorothy Graham, "Experiences of Automatisation de test: Case Studies of Software Automatisation de test", Addison-Wesley, 2012
[Fewster&Graham99]	Mark Fewster, Dorothy Graham, "Software Automatisation de test: Effective use of test execution tools", ACM Press Books, 1999, ISBN-10: 0201331403, ISBN-13: 9780201331400
[McCaffrey06]	James D. McCaffrey, ".NET Automatisation de test Recipes: A Problem-Solution Approach", APRESS, 2006 ISBN-13:978-1-59059-663-3, ISBN-10:1-59059-663-3

[Mosley02] Daniel J. Mosley, Bruce A. Posey, “Just Enough Software Automatisation de test”, Prentice Hall, 2002, ISBN-10: 0130084689, ISBN-13: 9780130084682

[Willcock11] Colin Willcock, Thomas Deiß, Stephan Tobies et Stefan Keil, “An Introduction to TTCN-3” Wiley, 2ème édition 2011, ISBN-10: 0470663065, ISBN-13: 978-0470663066

9.5 Références WEB

Identifiant	Référence de l’ouvrage
ISTQB-Web	Site Web du CFTL et de l’ISTQB. Se référer aux sites Web pour obtenir les dernières versions de glossaire et de syllabi.

10 Note aux fournisseurs de formation

10.1 Temps de formation

A chaque chapitre de ce syllabus correspond un temps alloué en minutes. L'objectif de cela est à la fois de fournir un guide sur la proportion de temps qui doit être allouée à chaque section d'un cours accrédité et de fournir un temps minimum approprié pour enseigner chaque section.

Les fournisseurs de formation peuvent passer plus de temps qu'indiqué et les candidats peuvent passer encore plus de temps en lecture et recherche. Un curriculum de cours n'a pas le même ordre que le syllabus. Il n'est pas demandé de conduire le cours dans un bloc continu de temps.

La table ci-dessous fournit un guide des temps pour enseigner et faire les exercices pour chaque chapitre (tous les temps sont donnés en minutes).

Chapitre	Minutes
0. Introduction	0
1. Introduction et objectifs de l'automatisation de test	30
2. Préparer l'automatisation de test	165
3. L'architecture générique d' automatisation de test	270
4. Risques de déploiement et contingences	150
5. Reporting et métriques d'automatisation de test	165
6. Transition des tests manuels vers un environnement automatisé	120
7. Vérifier la TAS	120
8. Amélioration continue	150
Total:	1170

La durée totale de cours en jours, basée sur une moyenne de sept heures par jour, est:
2 jours, 5 heures, 30 minutes.

10.2 Exercices pratiques sur le lieu de travail

Aucun n'exercice n'est conçu pour être exécuté sur le lieu de travail.

10.3 Règles pour le e-Learning

Toutes les parties de ce syllabus sont considérées appropriées pour être implémentées en e-learning.

11 Index