Certificazione di Tester

Syllabus Livello 'Advanced'

Test Automation – Engineering

Versione 2016

International Software Testing Qualifications Board

Copyright Notice
This document may be copied in its entirety, or extracts made, if the source is acknowledged.

Copyright © International Software Testing Qualifications Board (hereinafter called ISTQB®).

Gruppo di lavoro per il livello avanzato Test Automation: Bryan Bakker, Graham Bath, Armin Born, Mark Fewster, Jani Haukinen, Judy McKay, Andrew Pollner, Raluca Popescu, Ina Schieferdecker; 2016.

© International Software Testing Qualifications Board

Storia delle revisioni

Version	Date	Remarks
Initial Draft	13AUG2015	Initial draft
Second Draft	05NOV2015	LO mapping and repositioning
Third Draft	17DEC2015	Refined LOs
Beta Draft	11JAN2016	Edited draft
Beta	18MAR2016	Beta Release
Syllabus 2016	21OCT2016	GA Release

Storico delle Revisioni di traduzione per ITA-STQB

REV.	AUTORI	DESCRIZIONE DELLE MODIFICHE	DATA DI APPROVAZIONE DEL COMITATO SCIENTIFICO
01	Massimo Di Carlo	Traduzione del Syllabus TAE Advanced level	
01	Alessandro Collino, Marco Sogliani	Revisione della traduzione	28/4/2018
02	Massimo Di Carlo, Marco Sogliani	Alcune correzioni grammaticali	01/02/2019

© International Software Testing Qualifications Board

Indice dei contenuti

Storia delle revisioni 3
Indice dei contenuti....4
Ringraziamenti 6
1. Introduzione a questo Syllabus 7
1.1 Scopo di questo Documento 7
1.2 Panoramica 7
1.2.1 In Ambito 7
1.2.2 Fuori ambito....7
1.3 Il Tester certificato a livello Advanced come Test Automation – Engineer 8
1.3.1 Attese 8
1.3.2 Requisiti di ingresso e di rinnovo.... 8
1.3.3 Livello di conoscenza....8
1.3.4 Esami 9
1.3.5 Accreditamento....9
1.4 Parti Normative e Informative....9
1.5 Livello di dettaglio....9
1.6 Come è organizzato questo Syllabus....9
1.7 Termini, Definizioni e Acronimi.... 10
2. Introduzione e Obiettivi per l'Automazione del Testing - 30 min.... 11
2.1 Scopo dell'Automazione del Testing.... 12
2.2 Fattori di Successo nell'Automazione del Testing.... 13
3. Preparazione per l'Automazione del Testing - 165 min.... 16
3.1 Aspetti del SUT che influenzano l'Automazione del Testing 17
3.2 Valutazione e Selezione degli Strumenti.... 18
3.3 Progettare per la Testabilità e l'Automazione 20
4. La Generic Test Automation Architecture - 270 min.... 22
4.1 Introduzione alla gTAA.... 23
4.1.1 Panoramica sulla gTAA.... 24
4.1.2 Layer di Generazione del Testing.... 26
4.1.3 Layer di Definizione del Testing 27
4.1.4 Layer di Esecuzione del Testing.... 27
4.1.5 Layer di Adattamento del Testing.... 27
4.1.6 Gestione della Configurazione di una TAS.... 28
4.1.7 Gestione del Progetto di una TAS 28
4.1.8 Supporto della TAS per la Gestione del Testing.... 28
4.2 Progettazione di una TAA.... 29
4.2.1 Introduzione alla Progettazione di una TAA 29
4.2.2 Approcci per Automatizzare i Casi di Test.... 32
4.2.3 Considerazioni Tecniche sul SUT 38
4.2.4 Considerazioni sui Processi di Sviluppo/QA.... 39
4.3 Sviluppo di una TAS 40
4.3.1 Introduzione allo Sviluppo di una TAS 40

© International Software Testing Qualifications Board

4.3.2 Compatibilità tra TAS e SUT ... 41
4.3.3 Sincronizzazione tra TAS e SUT ... 41
4.3.4 Realizzare il Riuso nella TAS ... 43
4.3.5 Supporto per una varietà di sistemi target ... 44
5. Rischi di Deployment e Contingenze - 150 min. ... 45
5.1 Selezione di un Approccio di Automazione del Testing e Pianificazione del Deployment/Rollout ... 46
5.1.1 Progetto Pilota ... 46
5.1.2 Deployment ... 47
5.1.3 Deployment della TAS all'interno del Ciclo di Vita del Software ... 48
5.2 Valutazione del Rischio e Strategie di Mitigazione ... 48
5.3 Manutenzione dell'Automazione del Testing ... 50
5.3.1 Tipi di Manutenzione ... 50
5.3.2 Ambito e Approccio ... 51
6. Report e Metriche per l'Automazione del Testing - 165 min. ... 53
6.1 Selezione di Metriche per la TAS ... 54
6.2 Implementazione delle Misure ... 58
6.3 Logging della TAS e del SUT ... 59
6.4 Reportistica dell'Automazione del Testing ... 60
7. Transizione dal Testing Manuale ad un Ambiente Automatizzato - 120 min. ... 62
7.1 Criteri per l'Automazione ... 63
7.2 Identificare i passi necessari ad implementare l'Automazione per il Testing di Regressione ... 68
7.3 Fattori da considerare quando si implementa l'Automazione per il Testing di Nuove Funzionalità ... 70
7.4 Fattori da considerare quando si implementa l'Automazione del Testing Confermativo ... 71
8. Verificare la TAS - 120 min. ... 72
8.1 Verificare i Componenti dell'Ambiente di Test Automatizzato ... 73
8.2 Verificare la Suite di Test Automatizzati ... 75
9. Miglioramento Continuo - 150 min. ... 77
9.1 Opzioni per Migliorare l'Automazione del Testing ... 78
9.2 Pianificare l'Implementazione dei Miglioramenti per l'Automazione del Testing ... 80
10. Riferimenti ... 82
10.1 Standard ... 82
10.2 Documenti ISTQB ... 83
10.3 Marchi registrati ... 83
10.4 Libri ... 83
10.5 Riferimenti Web ... 84
11. Nota per i Training Providers ... 85
11.1 Tempi di apprendimento ... 85
11.2 Esercizi ... 85
11.3 Regole per l'e-Learning ... 85

© International Software Testing Qualifications Board

Ringraziamenti

Questo documento è stato prodotto da un team dedicato appartenente al gruppo di lavoro dell'International Software Testing Qualifications Board Advanced Level.

Il team desidera ringraziare il gruppo dei revisori e tutte le organizzazioni nazionali per i suggerimenti e gli input.

Al momento in cui l'Advanced Level Syllabus è stato completato, il gruppo di lavoro Advanced Level - Test Automation era costituto dai seguenti membri: Bryan Bakker, Graham Bath (Advanced Level Working Group Chair), Armin Beer, Inga Birthe, Armin Born, Alessandro Collino, Massimo Di Carlo, Mark Fewster, Mieke Gevers, Jani Haukinen, Skule Johansen, Eli Margolin, Judy McKay (Advanced Level Working Group Vice Chair), Kateryna Nesmyelova, Mahantesh (Monty) Pattan, Andrew Pollner (Advanced Level Test Automation Chair), Raluca Popescu, Ioana Prundaru, Riccardo Rosci, Ina Schieferdecker, Gil Shekel, Chris Van Bael.

Il team che ha redatto questo syllabus: Andrew Pollner (Chair), Bryan Bakker, Armin Born, Mark Fewster, Jani Haukinen, Raluca Popescu, Ina Schieferdecker.

Le seguenti persone hanno partecipato alla revisione, commento e votazione di questo syllabus (in ordine alfabetico): 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.

Questo documento è stato formalmente rilasciato dall'Assemblea Generale dell'ISTQB® .

© International Software Testing Qualifications Board

0. Introduzione a questo Syllabus

0.1 Scopo di questo Documento

Questo Syllabus è la base per l'International Software Testing Qualification di Livello Avanzato (Advanced Level) per Test Automation - Engineering.
L'ISTQB® mette a disposizione questo syllabus con le seguenti modalità:

- Ai Membri del Board, per consentirne la traduzione nella loro lingua madre e per accreditare i fornitori della formazione. I Board nazionali possono adattare il syllabus alle loro particolari esigenze linguistiche e modificare i riferimenti bibliografici per adattarli alle loro pubblicazioni locali.
- Agli Enti Esaminatori, per consentire la traduzione delle domande d'esame nella loro lingua adattandole agli obiettivi di apprendimento di ogni modulo.
- Ai Fornitori della formazione, per consentire la produzione di materiale didattico e mettere a punto metodologie d'insegnamento appropriate.
- Ai Candidati alla certificazione, per consentire la preparazione all'esame (partecipando a un corso o in modo indipendente).
- Alla comunità che si occupa d'ingegneria dei sistemi e del software, per consentire di migliorare la professione di tester del software e dei sistemi e per fornire una base di riferimento per libri e articoli.

L'ISTQB® permette inoltre ad altre entità di usare questo syllabus per altre finalità, purché ne chiedano ed ottengano anticipatamente un permesso scritto.

0.2 Panoramica

1.2.1 In Ambito

Questo documento descrive le attività dei un Test Automation Engineer (TAE) nella progettazione, sviluppo e manutenzione di soluzioni di automazione del testing. Si concentra sui concetti, i metodi, gli strumenti e i processi per automatizzare test funzionali dinamici e le relazioni tra questi test con la gestione del testing, la gestione della configurazione, la gestione dei difetti, i processi di sviluppo del software e l'assicurazione di qualità.

I metodi descritti sono in genere applicabili per vari approcci al ciclo di vita del software (per es. agile, sequenziale, incrementale, iterativo) e di tipi di test (testing funzionale e non funzionale).

1.2.2 Fuori ambito

I seguenti aspetti sono fuori ambito del syllabus Test Automation – Engineer:

- Gestione del test, creazione automatizzata di specifiche di test e generazione automatica dei test.
- Attività del Test Automation manager (TAM) nella pianificazione, supervisione e adattamento dello sviluppo e dell'evoluzione di soluzioni di automazione del testing.
- Specifiche di automazione dei test non funzionali (per es. prestazionali).

© International Software Testing Qualifications Board

- Automazione di analisi statica (per es. analisi delle vulnerabilità) e strumenti per il test statico.
- Insegnamento dei metodi di ingegneria del software e di programmazione (per es. quali standard usare e quali competenze avere per realizzare una soluzione di automazione del testing).
- Insegnamento delle tecnologie software (per es. quali tecniche di scripting usare per implementare una soluzione di automazione del testing).
- Selezione di prodotti e servizi per il testing del software (per es. quali prodotti e servizi usare per una soluzione di automazione del testing).

0.3 Il Tester certificato a livello Advanced come Test Automation – Engineer

1.3.1 Attese

La certificazione a livello advanced si rivolge a persone che desiderano aumentare le conoscenze e le competenze acquisite al livello foundation e sviluppare ulteriormente la loro competenza in una o più aree. I moduli offerti a livello di Advanced Specialist coprono un'ampia gamma di argomenti di testing.

Un Test Automation Engineer è una persona che ha una vasta conoscenza del testing in generale e una conoscenza approfondita nell'area specialistica dell'automazione del testing.

Una conoscenza approfondita è definita come l'avere sufficienti conoscenze della teoria e della pratica dell'automazione del testing tale da essere capace di influenzare la direzione che un'organizzazione e/o un progetto prende quando si progettano, si sviluppano e si manutengono soluzioni di automazione del testing per i test funzionali.

Il documento Overview dei moduli al livello advanced [ISTQB_AL_Modules] descrive i risultati di business per questo modulo.

1.3.2 Requisiti di ingresso e di rinnovo

I criteri di ingresso generali per il livello avanzato sono descritti nel sito ISTQB [ISTQB-Web], sezione Advanced Level.

In aggiunta a questi criteri di ingresso generali, i candidati devono avere il certificato ISTQB a Foundation Level [ISTQB-CTFL].

1.3.3 Livello di conoscenza

Gli obiettivi di apprendimento per questo syllabus sono mostrati all'inizio di ogni capitolo per una chiara identificazione. Ogni argomento in questo syllabus sarà esaminato in accordo con gli obiettivi di apprendimento ad esso assegnati.

I livelli di conoscenza assegnati agli obiettivi di apprendimento (K-level) sono descritti sul sito web ITA-STQB [ITA-STQB-Web].

© International Software Testing Qualifications Board

1.3.4 Esami

L'esame per questa Certificazione a livello Advanced deve essere basato sul questo syllabus più il Syllabus a livello Foundation [ISTQB-FL]. Ogni argomento nel syllabus sarà esaminato in accordo agli obiettivi di apprendimento assegnati.

Il format dell'esame è descritto sul sito ISTQB [ISTQB-Web], sezione Advanced Level. Alcune informazioni utili per chi deve sostenere gli esami sono incluse sempre sul sito ISTQB [ISTQB-Web].

1.3.5 Accreditamento

Un ISTQB Member Board può accreditare training provider il cui materiale dei corsi segue questo syllabus.

Il sito web ISTQB [ISTQB-Web], sezione Advanced Level descrive le regole specifiche che si applicano ai training provider per l'accreditamento dei corsi.

0.4 Parti Normative e Informative

Le parti normative di questo Syllabus esaminabili sono:

- Obiettivi di apprendimento
- Termini

Il resto del Syllabus è informativo e raffina gli obiettivi di apprendimento.

0.5 Livello di dettaglio

Il livello di dettaglio in questo Syllabus consente insegnamento ed esami a livello internazionale, Per ottenere tale obiettivo, il Syllabus è composto di:

- Obiettivi di apprendimento per ciascuna area, che descrivono il risultato di apprendimento di conoscenze e di modi di ragionare che devono essere raggiunti (questi sono normativi)
- Una lista di informazioni da insegnare, inclusa una descrizione dei concetti chiave, fonti come letteratura e standard diffusi e riferimenti a fonti aggiuntive se richiesto (queste sono di tipo informativo)

Il contenuto del Syllabus non è una descrizione dell'intera area di conoscenza dell'ingegneria dell'automazione del testing; riflette il livello di dettaglio che deve essere coperto in un corso di formazione accreditato a livello Advanced.

0.6 Come è organizzato questo Syllabus

Ci sono otto capitoli principali. Il titolo del capitolo riporta il tempo richiesto per il capitolo. Per esempio:

3. Architettura Generica di automazione del testing 270 min.

Indica che per il Capitolo 3 è previsto che abbia un tempo di 270 minuti per insegnare il materiale del capitolo. Gli specifici obiettivi di apprendimento sono elencati all'inizio di ogni capitolo.

© International Software Testing Qualifications Board

0.7 Termini, Definizioni e Acronimi

Molti termini usati nella letteratura del software sono usati in modo interscambiabile. Le definizioni in questo Syllabus Advanced Level sono disponibili nel Glossario Standard dei termini usati nel Software Testing, pubblicato da ISTQB [ISTQB-Glossary].

Tutte le parole chiave elencate all'inizio di ogni capitolo in questo Advanced Level Syllabus sono definite nel [ISTQB-Glossary].

In questo documento sono usati i seguenti acronimi:

CLI Command Line Interface
EMTE Equivalent Manual Test Effort
gTAA Generic Test Automation Architecture (Fornisce un blueprint per la soluzione di automazione del testing)
GUI Graphical User Interface
SUT System under test, si veda anche test object
TAA Test Automation Architecture (una istanza della gTAA per definire l'architettura di una TAS)
TAE Test Automation Engineer (la persona che è responsabile della progettazione di una TAA, inclusa la implementazione della TAS risultante, della sua manutenzione ed evoluzione tecnologica)
TAF Test Automation Framework (l'ambiente richiesto per l'automazione del testing inclusi i test harness e gli artefatti come le librerie di test)
TAM Test Automation Manager (la persona responsabile della pianificazione e della supervisione dello sviluppo e dell'evoluzione di una TAS)
TAS Test Automation Solution (La realizzazione / implementazione di una TAA, inclusi i test harness e gli artefatti come le librerie di test)
UI User Interface

© International Software Testing Qualifications Board

1. Introduzione e Obiettivi per l'Automazione del Testing - 30 min.

Termini

API testing, CLI testing, GUI testing, sistema sotto test (System Under Test), architettura di automazione del testing (test automation architecture), infrastruttura di automazione del testing (test automation framework), strategia di automazione del testing, automazione del testing, script di test, testware

Obiettivi di Apprendimento per Introduzione e Obiettivi per l'Automazione del Testing

1.1 Scopo dell'Automazione del Testing

ALTA-E-1.1.1 (K2) Spiegare gli obiettivi, i vantaggi, gli svantaggi e le limitazioni dell'automazione del testing

1.2 Fattori di Successo nell'Automazione del Testing

ALTA-E-1.2.1 (K2) Identificare i fattori tecnici di successo di un progetto di automazione del testing

© International Software Testing Qualifications Board

1.1 Scopo dell'Automazione del Testing

Nel testing del software, l'automazione del testing (che include l'esecuzione automatica del test) comprende una o più delle seguenti attività:

- Uso di particolari strumenti software per controllare e impostare le precondizioni del test
- Esecuzione di test
- Comparazione dei risultati effettivi con quelli previsti

Una buona pratica consiste nel separare il software usato per il testing dal sistema sotto test (SUT) per minimizzare le interferenze. Ci sono eccezioni, per esempio i sistemi embedded per i quali il software di test deve essere installato sul SUT.

Ci si attende che l'automazione del testing aiuti ad eseguire molti casi di test ripetutamente e sempre allo stesso modo su versioni differenti del SUT e/o dell'ambiente. Ma l'automazione del testing è più che un meccanismo per eseguire una suite di test senza interazione umana. Essa comprende anche un processo di progettazione di testware, che include:

- Software
- Documentazione
- Casi di test
- Ambienti di test
- Dati di test

Il testware è necessario per attività di test che comprendono:

- L'implementazione dei casi di test automatizzati
- Il monitoraggio e il controllo dell'esecuzione dei casi di test automatizzati
- L'interpretazione, il reporting e il logging dei risultati del test automatizzato

L'automazione del testing ha diversi approcci per testare un SUT:

- Testare tramite le interfacce pubbliche di classi, moduli o librerie del SUT (API testing)
- Testare tramite l'interfaccia utente del SUT (GUI testing o CLI testing)
- Testare tramite un servizio o un protocollo

Gli obiettivi dell'automazione del testing comprendono:

- Migliorare l'efficienza del test
- Fornire una più ampia copertura funzionale
- Ridurre il costo totale del testing
- Eseguire test che non è possibile eseguire manualmente
- Ridurre la durata di esecuzione del testing
- Aumentare la frequenza dei test, riducendo il tempo richiesto per i cicli di test

I vantaggi dell'automazione del testing comprendono:

- Per ogni build può essere eseguito un numero maggiore di test
- Possibilità di creare test che non possono essere eseguiti manualmente (test real-time, remoti, in parallelo)
- I test possono essere più complessi

© International Software Testing Qualifications Board

- I test sono eseguiti più velocemente
- I test sono meno soggetti ad errori umani
- Uso più efficace ed efficiente delle risorse di testing
- Feedback più rapidi sulla qualità del software
- Migliore affidabilità del sistema testato (ripetibilità, consistenza)
- Migliore consistenza dei test

Gli svantaggi dell'automazione del testing includono:

- Sono necessari costi aggiuntivi
- Investimento iniziale per il setup della TAS
- Sono richieste tecnologie aggiuntive
- Il team deve avere competenze di sviluppo e automazione
- È richiesta una continua manutenzione della TAS
- Può distrarre dagli obiettivi del testing, ad esempio concentrarsi ad automatizzare i casi di test a spese dell'esecuzione dei test
- I test possono diventare più complessi
- Possono essere introdotti errori aggiuntivi dall'automazione.

Tra i limiti dell'automazione del testing abbiamo:

- Non tutti i test manuali possono essere automatizzati
- L'automazione può solo controllare risultati interpretabili in modo automatico
- L'automazione può solo controllare risultati che possono essere verificati da un oracolo del testing automatico
- Non sostituisce il testing esplorativo

1.2 Fattori di Successo nell'Automazione del Testing

I seguenti fattori di successo si applicano a progetti di automazione del testing che sono da tempo operativi e quindi si concentrano su aspetti che hanno impatto sul successo a lungo termine di un progetto. I fattori che influenzano il successo di progetti di automazione del testing allo stadio di progetto pilota non sono qui considerati.
I più importanti fattori di successo per l'automazione del testing includono:

Test Automation Architecture (TAA)

La Test Automation Architecture (TAA) è strettamente collegata all'architettura del prodotto software. Deve essere chiaro quali requisiti funzionali e non funzionali l'architettura deve supportare. Tipicamente questi saranno i requisiti più importanti.

Spesso la TAA è progetta per la manutenibilità, le prestazioni e la facilità di apprendimento (si veda ISO/IEC 25000:2014 per i dettagli di queste e altre caratteristiche non funzionali). È utile coinvolgere ingegneri del software che comprendano l'architettura del SUT.

Testabilità del SUT

Il SUT deve essere progettato per favorire la testabilità a supporto dell'automazione del testing. Nel caso di testing tramite la GUI, ciò potrebbe significare che il SUT debba disaccoppiare quanto possibile l'interazione della GUI e i dati dalla struttura dell'interfaccia grafica. Nel caso di testing

© International Software Testing Qualifications Board

tramite le API, ciò potrebbe significare che debbano essere esposte come pubbliche un numero maggiore di classi, moduli e interfacce da linea di comando in modo che possano essere testate.

Le parti testabili del SUT dovrebbero essere affrontate per prime. In generale, un fattore chiave di successo dell'automazione del testing consiste nella facilità di implementare gli script di test automatizzati. Con questo obiettivo in mente, e anche per effettuare una proof of concept di successo, il Test Automation Engineer (TAE) ha bisogno di indentificare i moduli e i componenti del SUT che sono facilmente testabili con l'automazione e iniziare da essi.

Strategia di automazione del testing

Una strategia di automazione del testing pratica e coerente dovrebbe indirizzare la manutenibilità e la consistenza del SUT.

Può non essere possibile applicare una strategia di automazione del testing nello stesso modo sia a parti vecchie che nuove del SUT. Quando si crea una strategia di automazione, vanno considerati costi, benefici e rischi nell'applicarla a parti differenti del codice.

Si deve prendere in considerazione di testare con test automatizzati sia tramite l'interfaccia utente che tramite le API per verificare la consistenza dei risultati.

Test Automation Framework (TAF)

Un Test Automation Framework (TAF) che sia facile da usare, ben documentato e mantenibile, supporta un approccio consistente nell'automatizzare i test.

Per creare un TAF facile da utilizzare e mantenere, è necessario fare quanto segue:

- Implementare strumenti di reportistica: i report dei test dovrebbero fornire informazioni (passato/fallito/errore/non eseguito/abortito, statistiche, ecc.) sulla qualità del SUT. I report dovrebbero fornire informazioni per i tester coinvolti, i test manager, gli sviluppatori, i project manager e le altre persone coinvolte al fine di ottenere un quadro d'insieme della qualità.
- Consentire un facile troubleshooting: in aggiunta all'esecuzione e al logging; il TAF deve facilitare il troubleshooting dei test falliti. Il test può fallire a causa di:
    - anomalie nel SUT
    - anomalie nella TAS
    - problemi con i test stessi o con l'ambiente di test.
- Considerare in modo appropriato l'ambiente di test: gli strumenti di test sono dipendenti dalla consistenza dell'ambiente di test. Per il testing automatizzato è necessario avere un ambiente di test dedicato. Se non c'è controllo dell'ambiente di test e dei dati di test, il setup dei test può non soddisfare i requisiti per l'esecuzione dei test e porterà probabilmente a produrre dei risultati di esecuzione falsi.
- Documentare i casi di test automatizzati: gli obiettivi dell'automazione del testing devono essere chiari, per esempio: quali parti dell'applicazione devono essere testate, a quale livello e quali attributi devono essere testati (funzionali e non funzionali). Ciò deve essere chiaramente descritto e documentato.
- Tracciare il test automatizzato: la TAF deve supportare il tracciamento per permettere al test automation engineer di tracciare i singoli passi dei casi di test.

© International Software Testing Qualifications Board

- Consentire una facile manutenzione: idealmente, i casi di test automatizzati dovrebbero essere facilmente manutenuti affinché la manutenzione non consumi una parte significativa dello sforzo di automazione del testing. In aggiunta, lo sforzo per la manutenzione deve essere proporzionale al tasso di cambiamenti subiti dal SUT. Per ottenere ciò, i casi di test devono essere facilmente analizzabili, modificabili ed espandibili. Inoltre, il riuso del testware automatizzato dovrebbe essere elevato per minimizzare il numero di oggetti che richiedono cambiamenti.
- Mantenere aggiornati i test automatizzati: quando il cambiamento o l'aggiunta di requisiti causano il fallimento di alcuni test o di un'intera suite di test, i test falliti non devono essere tolti dall'esecuzione, ma vanno corretti.
- Pianificare per facilitare il rilascio degli script: si deve far in modo che gli script di test automatizzati possano essere facilmente rilasciati, modificati e di nuovo rilasciati.
- Eliminare gli script se necessario: si deve fare in modo di essere sicuri che gli script di test automatizzati possano essere facilmente eliminati se non sono più utili o necessari.
- Monitorare e ripristinare il SUT: nella pratica, nell'eseguire continuamente un caso di test o un insieme di casi di test, il SUT deve essere monitorato continuamente. Se nel SUT si presenta un errore bloccante (come un crash), il TAF deve avere la capacità di ripristinare l'esecuzione, saltare il caso di test corrente e riprendere il testing con il caso di test successivo.

Il codice dell'automazione del testing può essere complesso da manutenere. Non è infrequente avere tanto codice di testing quanto codice del SUT. Questa è la ragione per la quale è di estrema importanza che il codice di test sia mantenibile. Tale aspetto dipende dai differenti strumenti di test in uso, dai differenti tipi di verifica usati e dai differenti artefatti del testware che devono essere mantenuti (come dati di input del test, oracoli del test, report del test).

Con queste considerazioni sulla manutenzione in mente, in aggiunta alle azioni importanti che devono essere fatte, ci sono altre che dovrebbero essere evitate, come le seguenti:

- Non creare codice che sia sensibile alle interfacce (cioè che sia influenzato dai cambiamenti nell'interfaccia grafica o in parti non essenziali delle API)
- Non creare un'automazione del testing che sia sensibile al cambiamento dei dati o abbia un'alta dipendenza da particolari valori dei dati (per es. dati di input di un test che dipendano da output di altri test).
- Non creare un ambiente per il test automatizzato che sia sensibile al contesto (per es. data e orario del sistema operativo, parametri di zona geografica del sistema operativo o contenuti di altre applicazioni). In questo caso è meglio usare degli stub in modo che l'ambiente possa essere controllato.

Più sono i fattori di successo che sono soddisfatti, più probabilmente il progetto di automazione del testing avrà successo. Non tutti i fattori sono richiesti e in pratica raramente tutti i fattori sono soddisfatti. Prima di iniziare un progetto di automazione del testing, è importante analizzare le possibilità di successo considerando i fattori soddisfatti e i fattori mancanti, tendendo presenti i rischi dell'approccio scelto e il contesto del progetto. Una volta che la TAA è realizzata, è importante investigare quali aspetti sono mancanti o hanno ancora bisogno di lavoro per essere migliorati.

© International Software Testing Qualifications Board

2. Preparazione per l'Automazione del Testing - 165 min.

Termini
testabilità, driver, livello di intrusione, stub, strumento di esecuzione del test, test hook, test automation manager

Obiettivi di apprendimento per la Preparazione per l'Automazione del Testing

2.1 Aspetti del SUT che influenzano l'Automazione del Testing
ALTA-E-2.1.1 (K4) Analizzare un sistema sotto test per determinare la soluzione di automazione appropriata

2.2 Valutazione e Selezione degli Strumenti
ALTA-E-2.2.1 (K4) Analizzare gli strumenti per l'automazione del testing per un dato progetto ed effettuare un resoconto sugli aspetti tecnici e sulle scelte

2.3 Progettare per la Testabilità e l'Automazione
ALTA-E-2.3.1 (K2) Comprendere i metodi per "progettare per la testabilità" e " progettare per la l'automazione del testing" applicabili al SUT

© International Software Testing Qualifications Board

2.1 Aspetti del SUT che influenzano l'Automazione del Testing

Quando si valuta il contesto del SUT e il suo ambiente, devono essere identificati gli aspetti che influenzano l'automazione del testing per determinare una soluzione appropriata. Questi aspetti possono comprendere:

- Interfacce del SUT

  I casi di test automatizzati invocano azioni sul SUT. Per questo, il SUT deve fornire delle interfacce tramite le quali il SUT può essere controllato. Ciò può essere fatto tramite controlli della UI, ma anche per mezzo di interfacce software più a basso livello. In aggiunta, alcuni casi di test possono essere eseguiti interfacciando il SUT a livello della comunicazione (per es. usando TCP/IP, USB o interfacce proprietarie di messaggistica).

  La decomposizione del SUT permette all'automazione del testing di interfacciarsi con il SUT a diversi livelli di test. È possibile automatizzare i test ad un livello specifico (ad esempio a livello di componente o di sistema), ma solo quando il SUT lo supporta in modo adeguato. Ad esempio, a livello di componente, può non esserci alcuna interfaccia utente che possa essere utilizzata per il test, ed è quindi necessario che siano disponibili delle interfacce software, possibilmente personalizzate (chiamate anche test hook).

- Software di terze parti

  Spesso il SUT non consiste solo di software realizzato internamente all'organizzazione, ma può anche includere software fornito da terze parti. In alcuni contesti, questo software deve essere testato e se l'automazione del testing è applicabile, può essere necessaria per esso una diversa soluzione di automazione del testing, ad esempio basata sull'uso di API.

- Livelli di intrusione

  Differenti approcci all'automazione del testing (che usano diversi strumenti) hanno differenti livelli di intrusione. Più grande è il numero di cambiamenti che è necessario apportare al SUT in modo specifico per l'automazione del testing, più alto è il livello di intrusione. L'uso di interfacce software dedicate richiede un alto livello di intrusione mentre l'uso di elementi già esistenti di UI presenta un più basso livello di intrusione. L'uso di componenti hardware del SUT (come tastiere, interruttori manuali, touchscreen, interfacce di comunicazione) hanno un livello di intrusione ancora più alto.

  Il problema con i livelli di intrusione più alti è il rischio di falsi allarmi. Il TAS può mostrare esiti negativi che possono esser dovuti al livello di intrusione imposto dai test, ma che non è probabile accadano durante il normale funzionamento del sistema. Testare con un alto livello di intrusione è di solito una soluzione più semplice per l'approccio all'automazione del testing.

- Differenti architetture del SUT

  Differenti architetture del SUT possono richiedere differenti soluzioni di automazione del testing. Un approccio differente è necessario per un SUT scritto in C++ con la tecnologia COM rispetto ad un SUT scritto con Python. È possibile che queste differenti architetture siano gestite dalla stessa strategia di automazione, ma ciò richiede una strategia ibrida che le supporti entrambe.

- Dimensione e complessità del SUT

  Devono essere considerate dimensione e complessità del SUT attuale e i piani per gli sviluppi futuri. Per un SUT piccolo e semplice, può non essere giustificato un approccio complesso e ultra

© International Software Testing Qualifications Board

flessibile. Può essere più adatto un approccio semplice. Al contrario, può non essere saggio implementare un approccio leggero e semplice per un SUT molto grande e complesso. Tuttavia a volte, può essere appropriato iniziare con approccio di ridotte dimensioni e semplice anche per un SUT complesso, ma solo come approccio temporaneo (si veda il Capitolo 3 per maggiori dettagli).

Diversi aspetti qui descritti sono noti (per es. dimensione e complessità, interfacce software) quando il SUT è già disponibile, ma il più delle volte lo sviluppo dell'automazione del testing deve iniziare prima che il SUT sia disponibile. Quando ciò accade devono essere ipotizzate diverse cose o il TAE può specificare quali interfacce software siano necessarie (si veda il paragrafo 2.3 per maggiori dettagli).

Anche quando il SUT non esiste ancora, si può iniziare la pianificazione dell'automazione del testing. Per esempio:

- Quando i requisiti (funzionali o non funzionali) sono noti, i candidati all'automazione possono essere selezionati tra tali requisiti e allo stesso tempo possono essere identificati i modi per testarli. La pianificazione dell'automazione può iniziare per tali canditati, inclusa l'identificazione dei requisiti per l'automazione e la scelta della strategia dell'automazione del testing.
- Quando l'architettura e la progettazione tecnica sono state sviluppate, può essere intrapresa la progettazione delle interfacce software necessarie a supportare il testing.

2.2 Valutazione e Selezione degli Strumenti

La principale responsabilità del processo di valutazione e selezione degli strumenti appartiene al Test Automation Manager (TAM). Comunque il TAE sarà coinvolto per fornire informazioni al TAM e condurre molte delle attività di valutazione e selezione. Il concetto di processo di valutazione e selezione è stato introdotto al Livello Foundation e maggiori dettagli di questo processo sono descritti nel Syllabus Advanced Level – Test Manager [ISTQB-AL-TM].

Il TAE sarà coinvolto in tutto il processo di valutazione e selezione degli strumenti, ma darà specifici contributi nell'eseguire le seguenti attività:

- Valutare la maturità organizzativa e identificare le opportunità per il supporto degli strumenti di test
- Valutare gli obiettivi appropriati per il supporto degli strumenti di test
- Identificare e raccogliere le informazioni sugli strumenti potenzialmente adatti
- Analizzare le informazioni sugli strumenti rispetto agli obiettivi e ai vincoli di progetto
- Stimare il rapporto costi – benefici sulla base di un solido business case
- Indicare lo strumento più appropriato
- Identificare la compatibilità dello strumento con le componenti del SUT

Gli strumenti di automazione del testing funzionale di frequente non possono soddisfare tutte le attese o le situazioni che possono incontrarsi in un progetto di automazione. Segue un insieme di esempi di questi tipi di problemi (ma non è sicuramente una lista completa):

Situazione	Esempio	Possibili soluzioni
L'interfaccia dello strumento non funziona con altri strumenti che sono già in uso	• Lo strumento di gestione del test è stato aggiornato e l'interfaccia di integrazione è cambiata	• Porre attenzione alle release note prima di ogni aggiornamento e, per le migrazioni maggiori, testare

© International Software Testing Qualifications Board

	• Le informazioni dal supporto di prevendita erano sbagliate e non tutti i dati possono essere trasferiti allo strumento di reportistica	prima di migrare in produzione • Cercare di ottenere una dimostrazione sul campo dello strumento usando il SUT reale • Cercare il supporto dal venditore e/o dai forum della comunità degli utenti
Alcune dipendenze del SUT sono cambiate e non sono più supportate dallo strumento di test	• La struttura di sviluppo ha effettuato l'aggiornamento alla più recente versione di Java	• Sincronizzare gli aggiornamenti tra gli ambienti di sviluppo e test e lo strumento di automazione
Un oggetto sulla GUI non può essere catturato	• L'oggetto è visibile ma lo strumento di automazione del testing non può interagire con esso	• Provare ad usare solo tecnologie od oggetti ben conosciuti durante lo sviluppo • Fare un progetto pilota prima di comprare uno strumento di automazione del test • Far definire agli sviluppatori degli standard per gli oggetti
Lo strumento appare molto complicato	• Lo strumento ha un grande insieme di funzionalità, ma solo una parte sarà usata	• Cercare di trovare un modo per limitare l'insieme delle funzionalità, rimuovendo dalla barra degli strumenti quelle non desiderate • Scegliere una licenza che soddisfi le proprie esigenze • Cercare di trovare strumenti alternativi che siano più mirati alle funzionalità richieste
Conflitti con altri sistemi	• Dopo l'installazione di altri software lo strumento di automazione del testing non funziona più o viceversa	• Leggere le release note o i requisiti tecnici prima di installare • Ottenere conferma dal fornitore che non ci sarà alcun impatto sugli altri strumenti • Domandare ai forum delle comunità degli utenti
Impatto sul SUT	• Durante o dopo l'uso dello strumento di automazione del testing il SUT reagisce in modo diverso (per es. tempi di risposta più lunghi)	• Usare uno strumento che non abbia bisogno di cambiare il SUT (per es. installazione di librerie, ecc.)
Accesso al codice	• Lo strumento di automazione del testing cambierà parti del codice sorgente	• Usare uno strumento che non abbia bisogno di cambiare il codice sorgente (per es.

© International Software Testing Qualifications Board

		installazione di librerie, ecc.)
Risorse limitate (principalmente in ambienti embedded)	• L'ambiente di test ha limitate risorse libere o esaurisce le risorse (per es. la memoria)	• Leggere le release note e discutere dell'ambiente con il fornitore dello strumento per ottenere conferma che non porterà a tali problemi. • Domandare ai forum delle comunità degli utenti
Aggiornamenti	• Un aggiornamento non migra tutti i dati o corrompe gli script di test automatizzati, i dati o le configurazioni esistenti. • Un aggiornamento necessita di un ambiente differente (migliore)	• Testare l'aggiornamento sull'ambiente di test e ottenere conferma dal fornitore che la migrazione funzionerà • Leggere i prerequisiti dell'aggiornamento e decidere se l'aggiornamento vale lo sforzo • Cercare aiuto dai forum delle comunità degli utenti
Sicurezza	• Lo strumento di automazione del testing richiede informazioni che non sono disponili per il test automation engineer	• Il test automation engineer ha bisogno di ricevere le autorizzazioni necessarie
Incompatibilità tra differenti ambienti e piattaforme	• L'automazione del testing non funziona su tutti gli ambienti o piattaforme	• Implementare test automatizzati per massimizzare l'indipendenza dallo strumento minimizzando quindi il costo di usare più strumenti

2.3 Progettare per la Testabilità e l'Automazione

La testabilità del SUT (disponibilità di interfacce software che supportino il testing, per es. che rendano possibile il controllo e l'osservabilità del SUT) dovrebbe essere progettata e implementata in parallelo con la progettazione e l'implementazione delle altre funzionalità del SUT. Ciò può essere fatto dall'architetto del software (poiché la testabilità è proprio uno dei requisiti non funzionali del sistema), ma spesso è fatto da, o con il coinvolgimento di, un TAE.

La progettazione per la testabilità consiste di diverse parti:

- Osservabilità: è necessario che il SUT fornisca interfacce che diano visibilità all'interno del sistema. I casi di test possono allora usare queste interfacce per controllare, per esempio, se il comportamento atteso corrisponde a quello effettivo.
- Controllabilità: è necessario che il SUT fornisca interfacce che possano essere usate per effettuare azioni sul SUT. Queste possono essere componenti di UI, chiamate a funzioni, elementi di comunicazione (per es. protocollo TCP/IP o USB), segnali elettronici (per interruttori fisici), ecc.

© International Software Testing Qualifications Board

- Architettura chiaramente definita: la terza parte importante della testabilità è un'architettura che fornisca interfacce chiare e comprensibili, che diano controllo e visibilità a tutti I livelli di test.

Il TAE valuta i modi in cui il SUT può essere testato, compreso il testing automatizzato, in modo efficace (testare le giuste aree e trovare i bachi critici) ed efficiente (senza impiegare troppo sforzo). Ogni volta che siano necessarie specifiche interfacce software, esse devono essere specificate dal TAE e implementate dallo sviluppatore. È importante definire la testabilità e, se necessario, interfacce software aggiuntive quanto prima nel progetto, in modo che il lavoro di sviluppo possa essere pianificato e finanziato.

Alcuni esempi di interfacce software che supportano il test comprendono:

- Le potenti possibilità di scripting dei moderni fogli di calcolo.
- Applicare stub o mock per simulare software e/o hardware (per es. transazioni finanziare elettroniche, servizi software, server dedicati, schede elettroniche, parti meccaniche) che non è ancora disponibile o è troppo costoso da comprare, consente di testare il software in assenza di tale specifica interfaccia.
- Interfacce software (o stub e driver) possono essere usate per testare condizioni di errore. Si consideri un device con un hard disk drive interno (HDD), Il software che controlla questo HDD (chiamato driver) deve essere testato per esiti negativi o per l'usura dell'HDD. Fare ciò attendendo che l'HDD si guasti non è molto efficiente (o realizzabile). Implementando le interfacce software che simulano HDD difettosi o lenti si può verificare che il driver software si comporti correttamente (per es. fornendo messaggi di errore, nuovi tentativi).
- Interfacce software alternative possono essere usate per testare un SUT quando nessuna UI è ancora disponibile (e questo è spesso considerato in ogni caso un approccio raccomandabile). Il software embedded in sistemi tecnici spesso ha bisogno di monitorare la temperatura di un dispositivo in modo da far partire una funzione di raffreddamento quando la temperatura sale oltre un certo livello. Questo potrebbe essere testato senza hardware usando un'interfaccia software per specificare la temperatura.
- Il testing delle transazioni di stato è usato per valutare il comportamento al variare degli stati del SUT. Un modo per controllare se il SUT è nello stato corretto è interrogarlo tramite un'interfaccia software progettata appositamente per tale obiettivo (sebbene ciò includa un rischio, si veda il livello di intrusione nel paragrafo 2.1).

La progettazione per l'automazione dovrebbe considerare che:

- La compatibilità con gli strumenti di test esistenti deve essere accertata quanto prima.
- Il tema della compatibilità dello strumento di test è critico in quanto può impattare la possibilità di automatizzare test di funzionalità importanti (per es. l'incompatibilità con un oggetto griglia impedisce tutti i test che usano tale oggetto).
- Alcune soluzioni possono richiedere lo sviluppo di codice o chiamate ad API.

Progettare per la testabilità è di estrema importanza per un buon approccio all'automazione del testing e può anche portare beneficio all'esecuzione manuale del test.

© International Software Testing Qualifications Board

3. La Generic Test Automation Architecture - 270 min.

Termini

cattura/riesecuzione, testing guidato dai dati (testing data-driven), architettura generica di automazione del testing (generic test automation architecture), testing guidato dalle parole chiave (testing keyword-driven), scripting lineare, testing model-based, scripting guidato dal processo (scripting process-driven), scripting strutturato, layer di adattamento del testing, architettura di automazione del testing (test automation architecture), infrastruttura di automazione del testing (test automation framework), soluzione di automazione del testing (test automation solution), layer di definizione del testing, layer di esecuzione del testing, layer di generazione del testing

Obiettivi di apprendimento per la generic Test Automation Architecture

3.1 Introduzione alla gTAA

ALTA-E-3.1.1 (K2) Spiegare la struttura della gTAA

3.2 Progettazione di una TAA

ALTA-E-3.2.1 (K4) Progettare la TAA appropriata per un dato progetto
ALTA-E-3.2.2 (K2) Spiegare il ruolo che i layer svolgono all'interno di una TAA
ALTA-E-3.2.3 (K2) Comprendere le valutazioni per la progettazione di una TAA
ALTA-E-3.2.4 (K4) Analizzare i fattori di implementazione, uso e i requisiti di manutenzione per una data TAS

3.3 Sviluppo di una TAS

ALTA-E-3.3.1 (K3) Applicare le componenti di una generica TAA (gTAA) per costruire una TAA per un obiettivo specifico
ALTA-E-3.3.2 (K2) Spiegare i fattori da considerare quando si identifica la riusabilità dei componenti

© International Software Testing Qualifications Board

3.1 Introduzione alla gTAA

Un test automation engineer (TAE) ha il ruolo di progettare, sviluppare, realizzare e manutenere delle test automation solution (TAS). Quando si sviluppa una soluzione, è necessario svolgere attività simili, rispondere a domande simili e affrontare problemi simili, nonché assegnare loro una priorità. Questi concetti, passi e approcci ricorrenti nell'automatizzare il testing divengono la base della generic Test Automation Architecture, indicata in breve con gTAA.

La gTAA presenta i layer, le componenti e le interfacce che sono poi ulteriormente rifinite in una TAA concreta per una particolare TAS. Ciò consente un approccio strutturato e modulare alla costruzione di una soluzione di automazione del testing tramite:

- La definizione dello spazio concettuale, dei layer, dei servizi e delle interfacce di una TAS per la realizzazione di TAS tramite componenti sviluppate in casa o esternamente
- Il supporto di componenti semplificate per uno sviluppo dell'automazione del testing efficace ed efficiente
- Il riuso di componenti di testing automatizzato per TAS differenti o per TAS che si devono evolvere per linee e famiglie di prodotti software e per varie tecnologie e strumenti software
- La semplificazione della manutenzione e dell'evoluzione di TAS
- La definizione delle caratteristiche essenziali per un utente di una TAS

Una TAS comprende sia l'ambiente di test (e i suoi artefatti) che le suite di test (un insieme di casi di test inclusi i dati). Un test automation framework (TAF) può essere usato per realizzare una TAS. Esso fornisce il supporto per la realizzazione dell'ambiente di test e fornisce strumenti, test harness o librerie di supporto.

Si raccomanda che la TAA di una TAS rispetti i seguenti principi che aiutano ad effettuare facilmente sviluppo, evoluzione e manutenzione della TAS:

- Singola responsabilità: ogni componente della TAS deve avere una singola responsabilità e tale responsabilità deve essere incapsulata interamente nel componente. In altre parole, ogni componente di una TAS dovrebbe essere incaricato di esattamente una sola cosa, per es. generare keyword o dati, creare scenari di test, eseguire casi di test, effettuare il logging dei risultati, generare report di esecuzione.
- Estensione (si veda per es. open/closed principle di B. Myer): ogni componente di una TAS deve essere aperto all'estensione, ma chiuso per la modifica. Questo principio indica che dovrebbe essere possibile arricchire o modificare il comportamento dei componenti senza influenzare la compatibilità all'indietro delle funzionalità con la versione precedente.
- Sostituzione (Si veda per es. substitution principle di B. Liskov): ogni componente di una TAS deve essere sostituibile senza influenzare il comportamento generale della TAS. Il componente può essere rimpiazzato da uno o più componenti sostitutivi, ma il comportamento esposto deve essere lo stesso.
- Segregazione dei componenti (si veda per es. interfaces segregation principle di R.C. Martin): è meglio avere più componenti specifici che un componente generale multiuso. Ciò rende la sostituzione e la manutenzione più facile, eliminando dipendenze non necessarie.
- Inversione della dipendenza: i componenti di una TAS devono dipendere da astrazioni piuttosto che da dettagli di basso livello. In altre parole, i componenti non devono dipendere da specifici scenari di test automatizzati.

Tipicamente, una TAS basata sulla gTAA sarà implementata da un insieme di strumenti, i loro plugin e/o componenti. È importante notare che una gTAA è vendor-neutral: non definisce in anticipo nessun metodo concreto, tecnologia o strumento per la realizzazione della TAS. La gTAA può essere implementata con ogni approccio di ingegneria del software, per es. strutturato, object-oriented,

© International Software Testing Qualifications Board

service- oriented, model-driven, come anche da ogni tecnologia software e ogni strumento. Infatti, una TAS è spesso implementata usando strumenti off-the-shelf, ma saranno tipicamente necessarie aggiunte e/o adattamenti specifici per il SUT.

Altre linee guida e modelli di riferimento relativi alle TAS sono gli standard di ingegneria del software per i SDLC selezionati (Software Development Lifecycle), le tecnologie di programmazione, gli standard di formato, ecc. Non è tra gli obiettivi di questo Syllabus insegnare l'ingegneria del software in generale, comunque, da un TAE ci si attende che abbia capacità, esperienza e competenza nell'ingegneria del software.

Inoltre, un TAE ha bisogno di essere a conoscenza degli standard e delle best practice industriali di codifica e documentazione per farne uso durante lo sviluppo di una TAS. Queste pratiche possono incrementare la manutenibilità, l'affidabilità e la sicurezza della TAS. Tali standard sono tipicamente specifici del dominio. Standard popolari includono:

- MISRA per C o C++
- Standard di codifica JSF per C++
- Regole AUTOSAR per MathWorks Matlab/Simulink®

3.1.1 Panoramica sulla gTAA

La gTAA è strutturata in layer orizzontali come segue:

- Generazione del Testing
- Definizione del Testing
- Esecuzione del Testing
- Adattamento del Testing

La gTAA (si veda Figura 1: La Generic Test Automation Architecture) comprende quanto segue:

- Il layer di generazione del testing che supporta la progettazione manuale o automatica dei casi di test. Fornisce i mezzi per progettare i casi di test.
- Il layer di definizione del testing che supporta la definizione e l'implementazione delle suite di test e/o dei casi di test. Separa la definizione dei test dal SUT e/o da tecnologie e strumenti del sistema di test. Contiene i mezzi per definire test di alto livello e di basso livello, che sono composti da dati di test, casi di test, procedure di test e componenti di librerie di test o combinazioni di essi.
- Il layer di esecuzione del testing che supporta l'esecuzione dei casi di test e il logging dei test. Fornisce uno strumento di esecuzione del test per eseguire automaticamente i test selezionati e un componente di logging e reportistica.
- Il layer di adattamento del testing che fornisce il codice necessario per adattare i test automatizzati alle varie componenti ed interfacce del SUT. Fornisce differenti adattatori per connettersi al SUT tramite API, protocolli, servizi e altro.
- Ha anche interfacce per la gestione del progetto, la gestione della configurazione e la gestione del testing in relazione all'automazione del testing. Per esempio, l'interfaccia tra la gestione del testing e il layer di adattamento del testing gestisce la selezione e la configurazione degli adattatori appropriati in relazione alla configurazione del testing scelta.

Le interfacce fra i layer della gTAA e i loro componenti sono tipicamente specifici e, quindi, non ulteriormente approfondite qui.

© International Software Testing Qualifications Board

È importante comprendere che questi layer possono essere presenti o assenti in ogni data TAS. Per esempio:

- Se l'esecuzione del testing deve essere automatizzata, è necessario utilizzare i layer di esecuzione del testing e di adattamento del testing. Essi non hanno bisogno di essere separati e possono essere realizzati insieme, per es. in framework di unit test.
- Se la definizione del testing deve essere automatizzata, è richiesto il layer di definizione del testing.
- Se la generazione del testing deve essere automatizzata, è richiesto il layer di generazione del testing.

Più spesso, si inizierà con l'implementazione di una TAS dal basso verso l'alto, ma possono essere utili anche altri approcci come la generazione automatica dei test per i test manuali. In generale è consigliato implementare la TAS in passi incrementali (per es. in sprint) al fine di usare la TAS quanto prima possibile e dimostrare il valore aggiunto della TAS. Sono anche raccomandate delle proof of concept come parte di un progetto di automazione del testing.

Ogni progetto di automazione del testing ha bisogno di essere compreso, avviato e gestito come un progetto di sviluppo software e richiede una gestione del progetto dedicata. La gestione del progetto per lo sviluppo di un TAF (cioè il supporto dell'automazione del testing per un'intera organizzazione, per famiglie o linee di prodotti) può essere separata dalla gestione del progetto per la TAS (cioè l'automazione del testing per un prodotto concreto).

© International Software Testing Qualifications Board

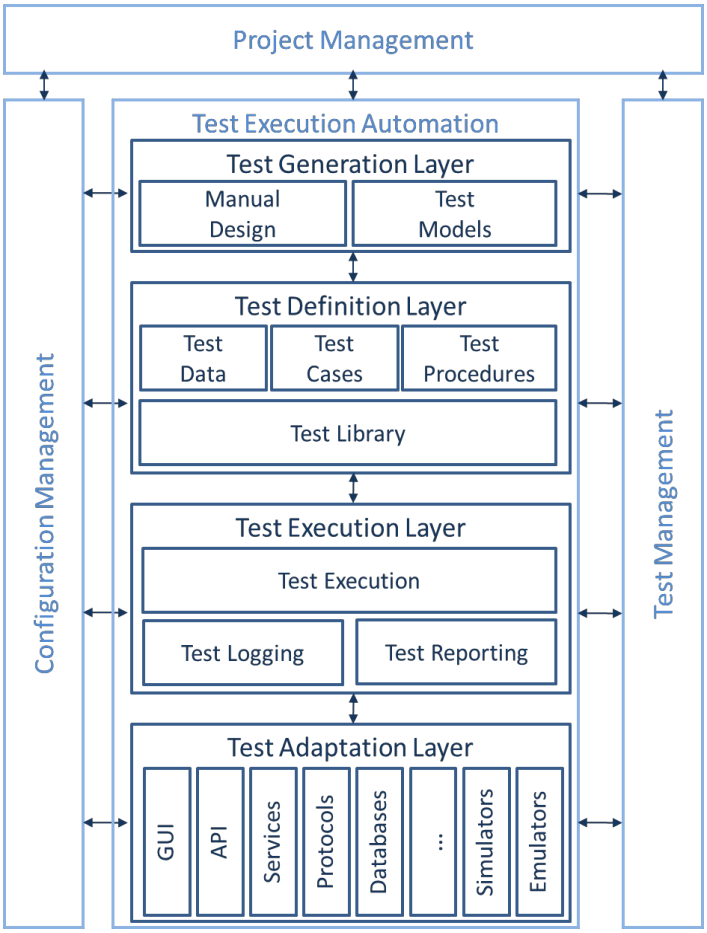

Figura 1: La Generic Test Automation Architecture

3.1.2 Layer di Generazione del Testing

Il layer di generazione del testing è costituito da strumenti di supporto per le seguenti attività:

- Progettazione manuale dei casi di test
- Creazione, cattura o derivazione di casi di test
- Generazione automatica di casi di test da modelli che definiscono il SUT e/o il suo ambiente (cioè model-based testing automatizzato)

I componenti in questo layer sono usati per:

- Editare e navigare le strutture di suite di test
- Mettere in relazione i casi di test con gli obiettivi del test o i requisiti del SUT
- Documentare la progettazione del testing

© International Software Testing Qualifications Board

Per la generazione automatica del testing si possono includere queste funzionalità:

- Abilità di modellare il SUT, il suo ambiente e/o il sistema di test
- Abilità di definire direttive di test e configurare/parametrizzare algoritmi di generazione dei test
- Abilità di tracciare all'indietro i test generati fino al modello (agli elementi)

3.1.3 Layer di Definizione del Testing

Il layer di definizione del testing è costituito da strumenti di supporto per le seguenti attività:

- Specificare i casi di test (ad alto e/o basso livello)
- Definire i dati di test per i casi di test di basso livello
- Specificare le procedure di test per un caso di test o un insieme di casi di test
- Definire gli script di test per l'esecuzione dei casi di test
- Fornire l'accesso alle librerie di test quando necessario (per esempio negli approcci keyword-driven)

I componenti in questo layer sono usati per:

- Partizionare/vincolare, parametrizzare o istanziare i dati di test
- Specificare sequenze di test o comportamenti completi dei test (inclusi passi di controllo ed espressioni), per parametrizzarli e/o per raggrupparli
- Documentare i dati di test, i casi di test e/o le procedure di test

3.1.4 Layer di Esecuzione del Testing

Il layer di esecuzione del testing è costituito da strumenti di supporto per le seguenti attività:

- Eseguire automaticamente i casi di test
- Effettuare il logging delle esecuzioni dei casi di test
- Effettuare il reporting dei risultati dei test

Il layer di esecuzione del testing può essere costituito da componenti che forniscono le seguenti funzionalità:

- Setup e teardown del SUT per l'esecuzione dei test
- Setup e teardown delle suite di test (cioè insieme di casi di test compresi i dati)
- Configurare e parametrizzare il setup dei test
- Interpretare sia i dati di test che i casi di test e trasformarli in script eseguibili
- Instrumentare il sistema di test e/o il SUT per il logging (filtrato) dell'esecuzione dei test e/o l'iniezione dei guasti
- Analizzare le risposte del SUT durante l'esecuzione dei test per indirizzare le esecuzioni dei test successivi
- Validare le risposte del SUT (comparazione dei risultati attesi ed effettivi) per i risultati delle esecuzioni dei casi di test automatizzati
- Controllare i tempi di esecuzione dei test automatizzati

3.1.5 Layer di Adattamento del Testing

Il layer di adattamento del testing è costituito da strumenti di supporto per le seguenti attività:

- Controllare la test harness

© International Software Testing Qualifications Board

- Interagire con il SUT
- Monitorare il SUT
- Simulare o emulare l'ambiente del SUT

Il layer di adattamento del testing fornisce le seguenti funzionalità:

- Mediare tra le definizioni dei test indipendenti dalla tecnologia e i requisiti della tecnologia specifica del SUT e degli strumenti di test
- Applicare gli adattatori specifici per le differenti tecnologie per interagire con il SUT
- Distribuire l'esecuzione dei test fra molteplici strumenti/interfacce di test o eseguire i test localmente

3.1.6 Gestione della Configurazione di una TAS

Normalmente una TAS è sviluppata in varie iterazioni/versioni e deve essere compatibile con le iterazioni/versioni del SUT. La gestione della configurazione di una TAS deve comprendere:

- Modelli di test
- Definizioni/specifiche dei test compresi i dati dei test, i casi di test e le librerie
- Script di test
- Motori di esecuzione dei test e componenti e strumenti aggiuntivi
- Adattatori di test per il SUT
- Simulatori ed emulatori per l'ambiente del SUT
- Risultati dei test e report dei test

Questi elementi costituiscono il testware e devono essere nella versione corretta che corrisponde alla versione del SUT. In alcune situazioni potrebbe essere necessario ritornare a versioni precedenti della TAS, per es. nel caso in cui sia necessario riprodurre problemi di esercizio con precedenti versioni del SUT. Una buona gestione della configurazione rende possibile tali attività.

3.1.7 Gestione del Progetto di una TAS

Poiché ogni progetto di automazione del testing è un progetto software, è richiesta la stessa gestione di qualsiasi altro progetto software. Un TAE, quando sviluppa una TAS, deve svolgere le attività per tutte le fasi della metodologia SDLC adottata. Inoltre, un TAE deve capire che l'ambiente di sviluppo della TAS deve essere progettato in modo che le informazioni sul suo stato (metriche) possano essere estratte facilmente o riportate automaticamente al project management della TAS.

3.1.8 Supporto della TAS per la Gestione del Testing

Una TAS deve supportare la gestione del testing per il SUT. I report dei test inclusi log di test e risultati dei test devono essere estratti facilmente o forniti automaticamente al test management (persone o sistemi) del SUT.

© International Software Testing Qualifications Board

3.2 Progettazione di una TAA

3.2.1 Introduzione alla Progettazione di una TAA

Ci sono delle attività principali richieste per progettare una TAA, che possono essere organizzate secondo le necessità del progetto di automazione del testing o dell'organizzazione. Queste attività sono discusse nei paragrafi seguenti. A seconda della complessità della TAA, può essere richiesto un numero maggiore o minore di attività.

Catturare i requisiti necessari per definire un'appropriata TAA

Tra i requisiti per un approccio all'automazione del testing si devono considerare i seguenti:

- Quale attività o fase del processo di testing deve essere automatizzata, per es. gestione del testing, progettazione dei test, generazione dei test o esecuzione dei test. Da notare che l'automazione del testing modifica il processo fondamentale del testing inserendo la generazione dei test tra la progettazione dei test e l'implementazione dei test.
- Quale livello di test deve essere supportato, per es. livello di componente, di integrazione, di sistema
- Quale tipo di test dovrebbe essere supportato, per es. testing funzionale, testing di conformità, testing di interoperabilità
- Quale ruolo di test dovrebbe essere supportato, per es. esecutore del test, analista del test, architetto del test, responsabile del test
- Quale prodotto software, linea di prodotti, famiglia di prodotti devono essere supportati, per es. per definire la durata e il tempo di vita della TAS implementata
- Quali tecnologie del SUT devono essere supportate, per es. definire la TAS in vista della compatibilità con le tecnologie del SUT

Confrontare e valutare differenti approcci di progettazione/architettura

Il TAE deve analizzare i pro e contro dei differenti approcci quando progetta i vari layer della TAA. Questi includono (senza escluderne altri):

Considerazioni sul layer di generazione del testing:

- Scelta tra la generazione manuale o automatica dei test.
- Scelta tra generazione basata – per esempio – sui requisiti, sui dati, sugli scenari o sul comportamento
- Scelta delle strategie di generazione dei test (per es. copertura del modello come alberi di classificazione per approcci basati sui dati, copertura dei casi d'uso, copertura delle eccezioni per approcci basati sugli scenari, copertura di transizioni/stati/cammini per approcci basati sul comportamento, ecc.)
- Scelta della strategia di selezione dei test. In pratica, la generazione combinatoria completa dei test non è fattibile, poiché può portare all'esplosione dei casi di test. Quindi, devono essere usati criteri pratici di copertura, pesi, valutazione dei rischi, ecc. per guidare la generazione e la successiva selezione dei test

Considerazioni sul layer di definizione del testing:

- Scelta tra definizione del testing data-driven, keyword-driven, pattern-based o model-driven
- Scelta della notazione per la definizione del testing (per es. tabelle, notazione basata su stati, notazione stocastica, notazione dei flussi dati, notazione dei processi di business,

© International Software Testing Qualifications Board

notazione basata su scenari, ecc. tramite l'uso di fogli di calcolo, linguaggi di test specifici del dominio, Testing and Test Control Notation (TTCN-3), UML Testing Profile (UTP), ecc.)
- Scelta delle linee guida e del formato per la definizione di test di alta qualità
- Scelta dei repository dei casi di test (fogli di calcolo, database, file, ecc.)

Considerazioni sul layer di esecuzione del testing:

- Scelta degli strumenti di esecuzione del test
- Scelta dell'approccio interpretato (con l'uso di una macchina virtuale) o compilato per implementare le procedure di test – questa scelta dipende tipicamente dalla scelta dello strumento di esecuzione del testing
- Scelta della tecnologia di implementazione per implementare le procedure di test (imperativa, come C; funzionale, come Haskell o Erlang; object-oriented, come C++, C#, Java; scripting, come Python o Ruby, o tecnologia specifica dello strumento) questa scelta è tipicamente dipendente dallo strumento scelto di esecuzione del test
- Scelta di librerie di supporto per facilitare l'esecuzione dei test (per es. librerie di dispositivi di test, librerie di codifica/decodifica, ecc.)

Considerazioni sul layer di adattamento del testing:

- Scelta delle interfacce di test con il SUT
- Scelta di strumenti per stimolare e osservare le interfacce di test
- Scelta degli strumenti per monitorare il SUT durante l'esecuzione dei test
- Scelta degli strumenti per tracciare l'esecuzione dei test (per es. includendo i tempi di esecuzione dei test)

Identificare le aree dove l'astrazione può portare benefici

L'astrazione in una TAA permette l'indipendenza dalla tecnologia nel senso che la medesima suite di test può essere usata in differenti ambienti di test e su differenti tecnologie. La portabilità degli artefatti del test aumenta. In aggiunta, si assicura la neutralità rispetto ai venditori, il che evita effetti di stretta dipendenza (lock-in) per la TAS. L'astrazione migliora anche manutenibilità e adattabilità al cambiamento o all'evoluzione nelle tecnologie del SUT. Inoltre, l'astrazione aiuta a rendere una TAA (e le sue implementazioni in TAS) più accessibili ai non tecnici, poiché le suite di test possono essere documentate (anche in modo grafico) e spiegate a più alto livello, il che migliora leggibilità e comprensione.

Il TAE deve discutere, con gli attori coinvolti nello sviluppo del software, nella quality assurance e nel testing, quale livello di astrazione usare in quale area della TAS. Per esempio, quali interfacce dei layer di adattamento e/o esecuzione del testing devono essere esternalizzate, definite formalmente e manutenute stabili per tutto il ciclo di vita della TAA? È anche necessario che si discuta se debba essere usata una definizione astratta del test o se la TAA debba usare soltanto un layer di esecuzione del testing con script di test. Ugualmente, c'è bisogno che si comprenda se la generazione del testing debba essere fatta in forma astratta con l'uso di modelli di test e approcci al model-based testing. Il TAE deve essere consapevole che ci sono dei compromessi fra implementazioni sofisticate e dirette di una TAA rispetto alla funzionalità complessiva, alla manutenibilità e all'espandibilità. Una decisione su quale livello di astrazione usare in una TAA deve tener conto di queste alternative.

Maggiore è il livello di astrazione usato per una TAA, maggiormente la TAA sarà flessibile rispetto ad ulteriori evoluzioni o transizioni a nuovi approcci o tecnologie. Ciò si raggiunge al costo di maggiori investimenti iniziali (per es. architetture e strumenti di automazione del testing più complessi, necessità di insieme di competenze più elevate, curve di apprendimento più ampie), il che ritarda il raggiungimento del punto di pareggio (breakeven), ma può essere più redditizio a lungo termine. Può anche portare a prestazioni ridotte della TAS.

© International Software Testing Qualifications Board

Mentre dettagliate considerazioni sul ROI (Return on Investment) sono di responsabilità del TAM, il TAE deve fornire dati per l'analisi del ROI tramite valutazioni tecniche e comparazioni tra differenti architetture e approcci alla automazione del testing, rispetto a tempi, costi, sforzi e benefici.

Comprendere le tecnologie del SUT e come queste si interconnettono con la TAS

L'accesso alle interfacce di test del SUT è centrale per ogni esecuzione dei test automatizzati. L'accesso può esser disponibile ai seguenti livelli:

- Livello software, per es. SUT e software di test sono linkati insieme
- Livello di API, per es. la TAS richiama funzioni/operazioni/metodi fornite da un'API (remota) del SUT
- Livello di protocollo, per es. la TAS interagisce con il SUT via HTTP, TCP, ecc.
- Livello di servizio, per es. la TAS interagisce con i servizi del SUT tramite servizi web, servizi RESTful, ecc.

In aggiunta, il TAE deve decidere sul paradigma di interazione della TAA da usare per l'interazione tra la TAS e il SUT, ogni volta che la TAS e il SUT siano separati da API, protocolli o servizi. Questi paradigmi includono i seguenti:

- Paradigma event-driven, che indirizza l'interazione tramite eventi che sono scambiati su un bus di eventi
- Paradigma Client-Server, che indirizza l'interazione tramite l'invocazione di servizi da un richiedente ad un fornitore di servizi
- Paradigma peer-to-peer, che indirizza l'interazione tramite l'invocazione di servizi da entrambi i sistemi

Spesso la scelta di un paradigma dipende dall'architettura del SUT e può avere implicazioni sull'architettura del SUT. L'interconnessione fra il SUT e la TAA deve essere analizzata accuratamente e progettata al fine di scegliere un'architettura fra i due sistemi a prova di futuro.

Comprendere l'ambiente del SUT

Un SUT può essere costituito da software standalone o da software che funziona solo in relazione ad altri software (per es. sistemi di sistemi), hardware (per es. sistemi embedded) o componenti di ambiente (per es. sistemi cyber-physical). Una TAS simula o emula l'ambiente del SUT come parte del setup del test automatizzato.

Esempi di ambienti di test e relativo uso includono:

- Un computer con sia il SUT che la TAS – utile per testare un'applicazione software
- Computer separati e connessi via rete per il SUT e la TAS – utile per testare software lato server
- Strumenti di test aggiuntivi per stimolare e osservare le interfacce tecniche di un SUT – utile per testare il software per esempio su un set-top box
- Strumenti di test connessi via rete per emulare l'ambiente operativo del SUT – utile per testare il software di un router di rete
- Simulatori per simulare l'ambiente fisico del SUT – utile per testare il software di un'unità di controllo embedded

Tempi e complessità per l'implementazione di una data architettura di testware

Mentre lo sforzo di stima per un progetto di una TAS è responsabilità del TAM, un TAE deve supportare il TAM in tale compito fornendo buone stime sul tempo necessario e sulla complessità del progetto di una TAA. Metodi per la stima e relativi esempi includono:

© International Software Testing Qualifications Board

- Stima basata sull'analogia come function point, three-point estimation, wideband delphi e expert estimation
- Stima tramite l'uso di work breakdown structure come quelle che si usano nel software o nei template di gestione dei progetti
- Stima parametrica come il Constructive Cost Model (COCOMO)
- Stime basate sulla dimensione come l'analisi dei Function Point, degli Story Point o degli Use Case
- Stime di gruppo come il metodo del Planning Poker

Facilità d'uso per una data implementazione di una architettura di testware

In aggiunta alle funzionalità della TAS, alla sua compatibilità con il SUT, alla sua stabilità a lungo termine e alla sua capacità di poter essere facilmente evoluta, ai suoi requisiti di sforzo e alle considerazioni sul ROI, un TAE ha la responsabilità specifica di indirizzare le esigenze di usabilità per una TAS. Ciò include, senza escludere altro:

- Progettazione orientata al tester
- Facilità di uso della TAS
- Supporto della TAS per altri ruoli nello sviluppo del software, nella quality assurance e nel project management
- Organizzazione, navigazione e ricerca efficaci nella TAS
- Documentazione, manuali e testi di help utili per la TAS
- Funzioni pratiche di reporting fornite dalla TAS e sulla TAS
- Progettazioni iterative per indirizzare feedback sulla TAS e richieste di modifica

3.2.2 Approcci per Automatizzare i Casi di Test

I casi di test devono essere tradotti in sequenze di azioni che sono eseguite sul SUT. Tali sequenze di azioni possono essere documentate in una procedura di test e/o possono essere implementate in script di test. Oltre alle azioni, i casi di test automatizzati devono anche definire i dati di test per l'interazione con il SUT e includono i passi di verifica per verificare che il risultato atteso sia stato ottenuto dal SUT. Possono essere usati vari approcci per creare la sequenza di azioni:

1. Il TAE implementa i casi di test direttamente all'interno degli script di test automatizzati. Questa opzione è la meno raccomandata per la mancanza di astrazione e l'incremento di carico per la manutenzione.
2. Il TAE progetta le procedure di test e le trasforma in script di test automatizzati. Questa opzione consente l'astrazione, ma manca dell'automazione per generare gli script di test.
3. Il TAE usa uno strumento per tradurre le procedure di test in script di test automatizzati. Questa opzione combina sia l'astrazione che la generazione automatica degli script.
4. Il TAE usa uno strumento per generare procedure di test automatizzate e/o traduce gli script di test direttamente da modelli. Questa opzione ha il più alto livello di automazione.

Da notare che le opzioni sono pesantemente dipendenti dal contesto del progetto. Può anche essere efficiente iniziare l'automazione del testing applicando una delle opzioni meno avanzate, poiché queste sono tipicamente più facili da implementare. Questo può fornire valore aggiunto breve termine sebbene porterà ad una soluzione meno mantenibile.

Approcci consolidati per automatizzare i casi di test comprendono:

- Approccio cattura/riesecuzione, che può essere usato per l'opzione 1

© International Software Testing Qualifications Board

- Approccio con scripting strutturato, approccio data-driven e approccio keyword-driven, che possono essere usati per le opzioni 2 o 3
- Testing model-based (incluso l'approccio process-driven), che può essere usato per l'opzione 4

Tali approcci sono spiegati di seguito in termini di concetti principali e pro e contro.

Approccio cattura/riesecuzione

Concetti principali

Negli approcci cattura / riesecuzione, si usano degli strumenti per catturare le interazioni con il SUT mentre si eseguono le sequenze di azioni come definite da una procedura di test. Gli input sono catturati; gli output possono anche essere registrati per verifiche successive. Durante la ripetizione degli eventi, ci sono varie possibilità manuali o automatiche di verifica degli output:

- Manuale: il tester deve controllare gli output del SUT per trovare eventuali anomalie
- Completo: tutti gli output del sistema che sono stati registrati durante la fase di cattura devono essere riprodotti dal SUT
- Esatto: tutti gli output del sistema che sono stati registrati durante la fase di cattura devono essere riprodotti dal SUT allo stesso livello di dettaglio di quanto è stato registrato
- Checkpoint: solo alcuni degli output sono verificati in certi punti per specifici valori

Pro

L'approccio cattura / riesecuzione può essere usato per SUT a livello di GUI e/o API. Inizialmente è facile da avviare ed usare.

Contro

Gli script creati con l'approccio cattura / riesecuzione sono difficili da manutenere e da evolvere perché l'esecuzione registrata dipende fortemente dalla versione del SUT su cui è stata effettuata la registrazione. Per esempio, quando si registra al livello di GUI, i cambiamenti al layout della GUI possono impattare gli script di test, anche se si tratta solo di un cambiamento nella posizione di un elemento della GUI. Perciò l'approccio cattura / riesecuzione rimane vulnerabile ai cambiamenti.

L'implementazione dei casi di test (script) può iniziare solo quando il SUT è disponibile.

Scripting lineare

Concetti principali

Come tutte le tecniche di scripting, lo scripting lineare parte da alcune procedure di test manuali. Da notare però che queste possono non essere documenti scritti – la conoscenza su quali test eseguire e come eseguirli può essere posseduta da uno o più Analisti del Test.

Ogni test è eseguito manualmente mentre lo strumento di test registra la sequenza delle azioni e in alcuni casi cattura gli output visibili del SUT su schermo. Questo generalmente produce come risultato uno script (di solito grande) per ciascuna procedura di test. Gli script registrati possono essere in seguito modificati per migliorare la leggibilità (per es. aggiungendo commenti in punti chiave per spiegare cosa sta succedendo) o per aggiungere ulteriori controlli usando il linguaggio di scripting dello strumento.

Gli script possono quindi essere rieseguiti dallo strumento, facendo ripetere allo strumento le stesse azioni eseguite dal tester quando lo script è stato registrato. Sebbene questa modalità

© International Software Testing Qualifications Board

possa essere usata per automatizzare test dalla GUI, non è una buona tecnica da utilizzare quando deve essere automatizzato un grande numero di test e tali test devono essere usati per molte release del software. Questo a causa dell'alto costo di manutenzione tipicamente causato dalle modifiche al SUT (ogni modifica al SUT può rendere necessari molti cambiamenti agli script registrati).

Pro

I vantaggi dello scripting lineare si concentrano sul fatto che è richiesto un lavoro di preparazione minimo o nullo prima che si possa iniziare ad automatizzare. Una volta che si è imparato ad usare lo strumento di automazione è solo una questione di registrare un test manuale e rieseguirlo (sebbene la parte di registrazione può richiedere un'interazione aggiuntiva con lo strumento di automazione per realizzare la comparazione tra i risultati reali e quelli attesi atta a verificare che il software stia funzionando correttamente). Non sono richieste capacità di programmazione anche se sono di solito utili.

Contro

Gli svantaggi degli script lineari sono numerosi. L'ammontare di lavoro richiesto per automatizzare ogni data procedura di test sarà per lo più dipendente dalla dimensione (numero di passi o azioni) richiesta per eseguirla. Così, la millesima procedura di test da automatizzare richiederà una quantità di lavoro simile in proporzione alla centesima procedura di test. In altre parole, non c'è molto spazio per diminuire il costo di realizzazione di nuovi test automatizzati.

Inoltre, se ci fosse un secondo script che eseguisse un test simile, sebbene con differenti valori di input, lo script conterrebbe la stessa sequenza di istruzioni del primo script; solo le informazioni contenute nelle istruzioni (argomenti o parametri delle istruzioni) sarebbero diverse. Se ci fossero vari test (e quindi script) essi conterrebbero tutti la stessa sequenza di istruzioni, ognuna delle quali dovrebbe essere manutenuta quando il software cambiasse in modo tale da influenzare gli script.

Poiché gli script sono realizzati in un linguaggio di programmazione, piuttosto che in linguaggio naturale, chi non è programmatore può trovare difficoltà nel comprenderli. Alcuni strumenti di test usano linguaggi proprietari (unici per lo strumento) che richiedono del tempo per essere appresi e diventare produttivi.

Gli script registrati contengono solo frasi generiche nei commenti, se presenti. Script lunghi in particolare sono meglio documentati, con commenti che spiegano cosa si sta facendo ad ogni passo del test. Ciò rende la manutenzione più facile. Gli script possono diventare presto molto grandi (contenendo molte istruzioni) quando il test comprende molti passi.

Gli script non sono modulari e difficili da manutenere. Lo scripting lineare non segue i comuni paradigmi di riusabilità e modularità del software ed è strettamente legato allo strumento che si sta usando.

Scripting strutturato

Concetti principali

La maggiore differenza tra la tecnica di scripting strutturato e la tecnica di scripting lineare è l'introduzione di una libreria di script. Questa contiene script riusabili che eseguono sequenze di istruzioni comunemente richieste da un certo numero di test. Buoni esempi di tali script sono quelli che si interfacciano, per es., con le operazioni delle interfacce del SUT.

Pro

© International Software Testing Qualifications Board

I benefici di tale approccio includono una riduzione significativa nelle modifiche richieste in manutenzione e un costo ridotto nell'automatizzare nuovi test (perché si possono usare script che già esistono piuttosto che doverli creare tutti da zero).

I vantaggi dello scripting strutturato sono largamente realizzati per mezzo del riuso degli script. Più test possono essere automatizzati senza dover creare il volume di script che un approccio di scripting lineare richiederebbe. Questo ha un impatto diretto sui costi di realizzazione e manutenzione. Il secondo test e i seguenti non richiederanno lo stesso sforzo per l'automazione perché potranno essere riusati di nuovo alcuni degli script creati per implementare il primo test.

Contro

Lo sforzo iniziale per creare gli script condivisi può essere visto come uno svantaggio, ma questo investimento iniziale si ripagherà abbondantemente se approcciato in modo appropriato. Saranno richieste capacità di programmazione per creare tutti gli script, perché solo una semplice registrazione non sarà sufficiente. La libreria di script deve essere ben gestita, cioè gli script devono essere documentati e deve essere facile per l'Analista Tecnico del Test trovare gli script richiesti (sarà qui d'aiuto una ragionevole convenzione sui nomi).

Testing Data-driven

Concetti principali

La tecnica di scripting data-driven è basata sulla tecnica di scripting strutturato. La differenza più significativa è come sono gestiti i dati di input. Gli input sono estratti dagli script e messi in uno o più file separati (chiamati tipicamente file di dati).

Ciò significa che lo script di test principale può essere riusato per implementare un certo numero di test (piuttosto che solo un singolo test). Tipicamente lo script principale 'riusabile' è chiamato lo script di 'controllo'. Lo script di controllo contiene la sequenza di istruzioni necessarie per eseguire i test, ma legge i dati di input da un file di dati. Un solo test di controllo può essere usato per molti test, ma è di solito insufficiente per automatizzare un ampio insieme di test. Quindi, sarà richiesto un certo numero di script di controllo, ma questo sarà solo una frazione del numero di test che sono automatizzati.

Pro

Il costo di aggiungere nuovi test automatizzati può essere ridotto significativamente da questa tecnica di scripting. Questa tecnica è usata per automatizzare molte variazioni di un test, fornendo un testing più approfondito in un'area specifica e quindi potendo aumentare la copertura del testing.

Avere dei test 'descritti' da file di dati comporta che l'Analista del Test può specificare test 'automatizzati' semplicemente popolando uno o più file di dati. Ciò dà all'Analista del Test più libertà di specificare test automatizzati senza troppa dipendenza dall'Analista Tecnico del Test (che può essere una risorsa scarsa).

Contro

Il bisogno di gestire i file di dati e assicurarsi che siano leggibili dalla TAS è uno svantaggio, ma può essere affrontato in modo appropriato.

Inoltre, possono essere persi alcuni importanti casi di test negativi. I test negativi sono una combinazione di procedure di test e dati di test. In un approccio che punta principalmente ai dati di test, è possibile che "le procedure di test negative" siano trascurate.

© International Software Testing Qualifications Board

Testing Keyword-driven

Concetti principali

La tecnica di scripting keyword-driven si basa sulla tecnica di scripting data-driven. Ci sono due principali differenze: (1) i file di dati sono ora chiamati file di 'definizione del test' o qualcosa di simile (per es. file delle action word); (2) c'è un solo script di controllo.

Un file di definizione del test contiene una descrizione dei test che sia facile da comprendere da parte dell'Analista del Test (più facile di un equivalente file di dati). Di solito conterrà dati come i file di dati, ma i file di keyword conterranno anche istruzioni ad alto livello (le keyword, o 'action word').

Le keyword devono essere scelte in modo che abbiano significato per l'Analista del Test, per i test descritti e per l'applicazione testata. Esse sono principalmente (ma non esclusivamente) usate per rappresentare interazioni di business di alto livello con un sistema (per es. "crea un ordine"). Ogni keyword rappresenta un certo numero di interazioni dettagliate con il sistema sotto test. Sequenze di keyword (compresi i dati di test necessari) sono usate per specificare i casi di test. Keyword speciali possono essere usate per i passi di verifica o le keyword possono contenere sia le azioni che i passi di verifica.

L'ambito di responsabilità dell'Analista del Test include la creazione e la manutenzione dei file di keyword. Ciò significa che una volta che gli script che le implementano sono realizzati, l'Analista del Test può aggiungere test 'automatizzati' semplicemente specificandoli in un file di keyword (come con lo scripting data-driven).

Pro

Una volta che lo script di controllo e gli script che realizzano le keyword sono stati scritti, il costo di aggiungere nuovi test automatizzati sarà molto ridotto da questa tecnica di scripting.

Avere i test 'descritti' da file di keyword significa che l'Analista del Test può specificare test 'automatizzati' semplicemente descrivendo i test tramite le keyword e i dati associati. Ciò rende l'Analista del Test più libero di specificare test automatizzati senza dipendere troppo dall'Analista Tecnico del Test (che può essere una risorsa scarsa). Il beneficio a questo riguardo dell'approccio keyword-driven rispetto all'approccio data-driven è dato dall'uso delle keyword. Ciascuna keyword deve rappresentare una sequenza dettagliata di azioni che produce qualche risultato significativo. Per esempio, 'creare un'utenza', 'inserire un ordine', 'controllare lo stato dell'ordine' sono tutte azioni possibili per un'applicazione di shopping online che comprendono un certo numero di passi dettagliati. Quando un Analista del Test descrive un test di sistema ad un altro Analista del Test, è probabile che essi parlino tra loro in termini di queste azioni ad alto livello, non dei passi dettagliati. L'obiettivo dell'approccio keyword-driven quindi è di implementare queste azioni ad alto livello e consentire che i test siano definiti in termini di azioni ad alto livello senza riferimento a passi dettagliati.

Questi casi di test sono più facili da manutenere, leggere e scrivere poiché la complessità può essere nascosta nelle keyword (o nelle librerie, nel caso di un approccio di scripting strutturato). Le keyword possono offrire un'astrazione dalle complessità delle interfacce del SUT.

Contro

Implementare le keyword rimane un impegno significativo per i test automation engineer, in particolare se si usa uno strumento che non offre alcun supporto per questa tecnica di scripting. Per piccoli sistemi può esserci troppo overhead per implementarle e i costi possono superare i benefici.

© International Software Testing Qualifications Board

È necessario prendere precauzioni per assicurarsi che siano implementate le keyword corrette. Delle buone keyword saranno usate spesso in molti test differenti, mentre keyword di scarsa qualità è probabile che siano usate solo per una o poche volte.

Scripting Process-driven

Concetti principali

L'approccio process-driven si basa sulla tecnica di scripting keyword-driven con la differenza che sono degli scenari – che rappresentano casi d'uso del SUT e loro varianti – a costituire gli script che sono parametrizzati con i dati di test o combinati in definizioni di test di più alto livello.

Tali definizioni di test sono più facili da trattare poiché può essere determinata la relazione logica fra le azioni, per es. 'verifica lo stato dell'ordine' dopo 'inserire un ordine' in un test positivo o 'verifica lo stato dell'ordine' senza aver usato in precedenza 'inserire un ordine' in un test di robustezza.

Pro

L'uso di definizioni di casi di test basate su scenari e processi consente la definizione delle procedure di test dal punto di vista del flusso operativo (workflow). L'obiettivo dell'approccio process-driven è implementare tali flussi operativi di alto livello usando librerie di test che rappresentano i passi di test dettagliati (si veda anche l'approccio keyword-driven).

Contro

Può non essere facile comprendere i processi di un SUT da parte di un Analista Tecnico del Test e così anche l'implementazione di script orientati ai processi, in particolare se nessuna logica dei processi di business è supportata dallo strumento di automazione.

Si deve fare attenzione anche di assicurarsi che siano implementati, per mezzo di keyword corrette, i processi corretti. Buoni processi saranno referenziati da altri processi e ciò risulterà in molti test rilevanti, mentre processi scadenti fatti non avranno un ritorno in termini di rilevanza, capacità di trovare gli errori, ecc.

Testing Model-based

Concetti principali

Il testing model-based si riferisce alla generazione automatica dei casi di test (si veda anche il Model-Based Tester Syllabus di ISTQB) – come opposta all'esecuzione automatica dei casi di test – con l'uso di cattura/riproduzione, scripting lineare, scripting strutturato, scripting data-driven, scripting keyword-driven o scripting process-driven. Il testing model-based usa modelli (semi-) formali che astraggono dalle tecnologie di scripting della TAA. Diversi metodi di generazione dei test possono essere usati per derivare test per ognuno dei framework di scripting discussi finora.

Pro

Il testing model-based consente con l'astrazione di concentrarsi sull'essenza del testing (in termini di logica di business, dati, scenari, configurazioni, ecc. da testare). Consente anche di generare test per differenti sistemi target e tecnologie, cosicché i modelli usati per la generazione del test costituiscono una rappresentazione a prova di futuro del testware, che può essere riusato e manutenuto quando la tecnologia evolve.

© International Software Testing Qualifications Board

In caso di cambiamenti nei requisiti, il modello di test deve essere solo adattato: viene generato automaticamente un insieme completo di casi di test. Le tecniche di progettazione dei casi di test sono incorporate nei generatori di casi di test.

Contro

È richiesta conoscenza nella modellazione per applicare un approccio di testing model-based in modo efficace. L'attività di modellare per astrazione le interfacce, i dati e/o il comportamento di un SUT può essere difficile. In aggiunta, la modellazione e gli strumenti di testing model-based non sono ancora popolari, ma stanno maturando. Gli approcci al testing model-based richiedono adattamenti dei processi di test. Per esempio, deve essere stabilito il ruolo del progettista del test. In aggiunta, i modelli usati per la generazione dei test costituiscono degli artefatti importanti per la quality assurance di un SUT, che devono essere verificati per la qualità e anche manutenuti.

3.2.3 Considerazioni Tecniche sul SUT

In aggiunta a quanto sopra, si devono considerare degli aspetti tecnologici di un SUT quando si progetta una TAA. Alcuni di questi sono discussi di seguito, sebbene questa non sia una lista completa e deve quindi servire come esempio di aspetti importanti.

Interfacce del SUT
Un SUT ha delle interfacce interne (dentro al sistema) ed interfacce esterne (verso l'ambiente del sistema e i suoi utenti o tramite componenti esposti). Una TAA deve essere capace di controllare e/o osservare tutte queste interfacce del SUT che sono potenzialmente toccate dalle procedure di test (cioè, le interfacce devono essere testabili). In aggiunta, ci può anche essere bisogno di registrare le interazioni fra il SUT e la TAS con differenti livelli di dettaglio, tipicamente includendo dei timestamp.

Attenzione al testing (per es. un test) è necessaria all'inizio del progetto (o continuamente negli ambienti agile) durante la definizione dell'architettura per verificare la disponibilità di interfacce di test o servizi per il test necessari affinché il SUT sia testabile (progettazione per la testabilità).

Dati del SUT
Un SUT usa dati di configurazione per controllare la sua istanziazione, configurazione, amministrazione, ecc. Inoltre, usa i dati utente che elabora. Un SUT può anche usare dati esterni da altri sistemi per completare le sue attività. A seconda delle procedure di test per un SUT, tutti questi tipi di dati devono essere definibili, configurabili e in grado di essere istanziati dalla TAA. Il modo specifico di gestire i dati del SUT è deciso durante il progetto della TAA. A seconda dell'approccio, i dati possono essere trattati come parametri, fogli di dati di test, database di test, dati reali, ecc.

Configurazioni del SUT
Un SUT può essere installato in configurazioni differenti, per esempio su diversi sistemi operativi, su differenti apparati o con differenti impostazioni di lingua. A seconda delle procedure di test, differenti configurazioni del SUT possono essere indirizzate dalla TAA. Le procedure di test possono richiedere differenti setup di test (in un laboratorio) o setup di test virtuali (in cloud) di una TAA in combinazione con una data configurazione del SUT. Può anche essere richiesto di aggiungere simulatori e/o emulatori di particolari componenti del SUT per particolari aspetti del SUT.

Standard e aspetti legali del SUT
In aggiunta gli aspetti tecnici di un SUT, nel progetto di una TAA può essere necessario rispettare requisiti legali e/o standard in modo da progettare la TAA in modo compatibile. Esempi includono requisiti

© International Software Testing Qualifications Board

di privacy per i dati di test o requisiti di riservatezza che impattano le funzionalità di logging e reporting della TAA.

Strumenti e ambienti usati per sviluppare il SUT
Durante lo sviluppo di un SUT, possono essere usati diversi strumenti per l'ingegnerizzazione dei requisiti, la progettazione e la modellazione, la codifica, l'integrazione e il deployment del SUT. La TAA insieme con i suoi stessi strumenti deve tener conto dell'intero panorama degli strumenti del SUT per consentire la compatibilità tra gli strumenti, la tracciabilità e/o il riuso degli artefatti.

Interfacce di test nel prodotto software
È fortemente raccomandato di non rimuovere tutte le interfacce di test prima del rilascio del prodotto. In molti casi, queste interfacce possono essere lasciate nel SUT senza causare problemi nel prodotto finale. Quando lasciate, le interfacce possono essere usate dagli ingegneri dell'esercizio e del supporto per la diagnosi dei problemi e per testare i successivi rilasci di manutenzione. È importante verificare che le interfacce non pongano alcun rischio di sicurezza. Se necessario, gli sviluppatori di solito possono disabilitare tali interfacce in modo tale che non possano essere usate al di fuori della struttura di sviluppo.

3.2.4 Considerazioni sui Processi di Sviluppo/QA

Devono essere considerati gli aspetti relativi ai processi di sviluppo e di quality assurance di un SUT quando si progetta una TAA. Alcuni di questi sono discussi di seguito, sebbene non si tratti di una lista completa e serva quindi solo a titolo di esempio di aspetti importanti.

Requisiti sul controllo dell'esecuzione dei test
A seconda del livello di automazione richiesto dalla TAA, può essere necessario che la TAA debba supportare l'esecuzione interattiva dei test, l'esecuzione batch o l'esecuzione completamente automatica.

Requisiti di reporting
A seconda dei requisiti di reporting, inclusi i tipi di report e le loro strutture, la TAA deve poter essere in grado di supportare report dei test fissi, parametrizzati o definiti in formati e layout differenti.

Ruoli e diritti di accesso
A seconda dei requisiti di sicurezza, alla TAA può essere richiesto di fornire un sistema di ruoli e di diritti di accesso.

Insieme degli strumenti utilizzati
La gestione del progetto del SUT, la gestione del testing, i repository del codice e dei test, il tracciamento dei difetti, la gestione degli incidenti, l'analisi dei rischi, ecc. possono essere tutti supportati da strumenti che compongono un insieme di strumenti stabilito. La TAA è anche supportata da uno strumento o da un gruppo di strumenti che è necessario integrare con gli altri strumenti nell'insieme. Inoltre, gli script di test devono essere memorizzati e versionati come il codice del SUT in modo tale che le revisioni seguano lo stesso processo per entrambi.

© International Software Testing Qualifications Board

3.3 Sviluppo di una TAS

3.3.1 Introduzione allo Sviluppo di una TAS

Lo sviluppo di una TAS è comparabile agli altri progetti di sviluppo software. Può seguire le stesse procedure e gli stessi processi inclusi le peer review da sviluppatori e tester. Specifiche di una TAS sono la sua compatibilità e sincronizzazione con il SUT. Queste richiedono considerazioni sulla progettazione della TAA (si veda il paragrafo 3.2) e sullo sviluppo della TAS. Inoltre, il SUT è impattato dalla strategia di test, per es., nel dover rendere le interfacce di test disponibili per la TAS.

Questo paragrafo usa il ciclo di vita dello sviluppo software (SDLC) per spiegare il processo di sviluppo della TAS e gli aspetti correlati di compatibilità e sincronizzazione con il SUT. Questi aspetti sono allo stesso modo importanti per ogni altro processo di sviluppo che sia stato scelto o sia in essere per lo sviluppo del SUT e/o della TAS – essi devono essere adattati di conseguenza.

Il SDLC di base per la TAS è mostrato nella Figura 2.

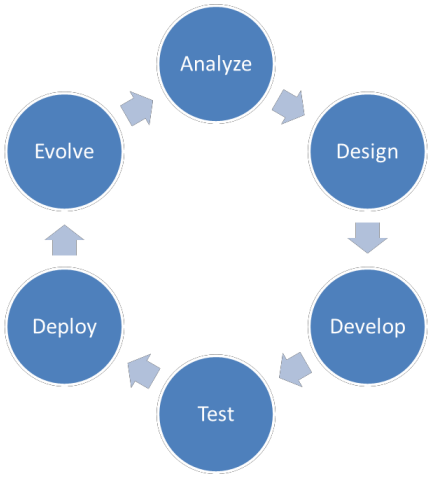

Figura 2: Un SDLC di base per una TAS

L'insieme dei requisiti per una TAS deve essere raccolto e analizzato (si veda la Figura 2). I requisiti guidano la progettazione della TAS come definita dalla sua TAA (si veda il paragrafo 3.2). La progettazione è convertita in software con gli approcci tipici dell'ingegneria del software. Si noti che una TAS può anche usare dispositivi hardware di test dedicati, che sono al di fuori degli argomenti trattati da questo Syllabus. Come ogni altro software, una TAS deve essere testata. Ciò è fatto tipicamente con test di base di verifica della funzionalità della TAS che sono seguiti da test sulla mutua interazione tra TAS e SUT. Dopo il deployment e l'uso della TAS, spesso è necessaria un'evoluzione della TAS per aggiungere altre funzionalità di test, cambiare i test o aggiornare la TAS per allinearla ai cambiamenti del SUT. L'evoluzione della TAS richiede un nuovo giro di sviluppo della TAS secondo il SDLC.

Si noti anche che il SDLC non mostra aspetti di backup, archiviazione e teardown di una TAS. Come con lo sviluppo della TAS, queste procedure devono seguire metodi stabiliti dalla struttura organizzativa.

© International Software Testing Qualifications Board

3.3.2 Compatibilità tra TAS e SUT

Compatibilità dei processi
Il testing di un SUT deve essere sincronizzato con il suo sviluppo e, in caso di automazione dei test, sincronizzato con lo sviluppo della TAS. Quindi, è vantaggioso coordinare i processi per lo sviluppo del SUT, per lo sviluppo della TAS e per il testing. Un notevole guadagno può essere ottenuto quando lo sviluppo di SUT e TAS sono compatibili in termini di struttura dei processi, gestione dei processi e strumenti di supporto.

Compatibilità dei team
La compatibilità dei team è un altro aspetto di compatibilità fra sviluppo di TAS e SUT. Se è usato un modo di ragionare compatibile per affrontare e gestire lo sviluppo della TAS e del SUT, entrambi i team beneficeranno della revisione reciproca dei requisiti, delle progettazioni e/o degli artefatti sviluppati, discutendo le problematiche e trovando soluzioni compatibili. La compatibilità dei team aiuta anche nella comunicazione e nell'interazione reciproca.

Compatibilità delle tecnologie
Inoltre, deve essere considerata la compatibilità tecnologica fra la TAS e il SUT. È buona cosa progettare e realizzare una mutua interazione senza interruzioni fra TAS e SUT sin dall'inizio. Anche se ciò non è possibile (per es. perché le soluzioni tecniche non sono disponibili per la TAS o il SUT), una mutua interazione senza interruzioni può essere possibile con l'uso di adattatori, wrapper o altre forme di intermediari.

Compatibilità degli strumenti
Deve essere presa in considerazione la compatibilità degli strumenti usati per la gestione, lo sviluppo e l'assicurazione della qualità di TAS e SUT. Per esempio, se sono usati gli stessi strumenti la gestione dei requisiti e/o la gestione dei difetti, lo scambio di informazioni e il coordinamento dello sviluppo di TAS e SUT saranno più facili.

3.3.3 Sincronizzazione tra TAS e SUT

Sincronizzazione dei requisiti
Dopo l'identificazione dei requisiti, devono essere sviluppati sia i requisiti del SUT che della TAS. I requisiti della TAS possono essere raggruppati in 2 gruppi principali: (1) requisiti che indirizzano lo sviluppo della TAS in quanto sistema software, come i requisiti per le funzionalità della TAS per la progettazione dei test, la specifica dei test, l'analisi dei risultati dei test, ecc. e (2) requisiti che indirizzano il testing del SUT per mezzo della TAS. Questi, chiamati requisiti del testing, corrispondono ai requisiti del SUT e riflettono tutte quelle funzionalità e proprietà del SUT che devono essere testate dalla TAS. Ogni volta che i requisiti del SUT o della TAS sono aggiornati, è importante verificare la consistenza tra i due e controllare che per tutti i requisiti del SUT che devono essere testati dalla TAS abbiano requisiti di test definiti.

Sincronizzazione delle fasi di sviluppo
Per avere la TAS pronta quando necessario per testare il SUT, le fasi di sviluppo devono essere coordinate. Si ha più efficienza quando le fasi di analisi dei requisiti, progettazione, specifica e implementazione del SUT e della TAS sono sincronizzate.

© International Software Testing Qualifications Board

Sincronizzazione del tracciamento dei difetti
I difetti possono essere collegati al SUT, alla TAS o a requisiti/progettazioni/specifiche. A causa delle relazioni fra i due progetti, ogni volta che si ha un difetto in uno, l'azione correttiva può avere impatto anche sull'altro. Il tracciamento dei difetti e i test di confermativi devono coinvolgere sia TAS che SUT.

Sincronizzazione delle evoluzioni di SUT e TAS
Sia il SUT che la TAS possono evolvere per supportare nuove funzionalità o rimuovere funzionalità esistenti, per correggere difetti o indirizzare cambiamenti nel loro ambiente (inclusi cambiamenti rispettivamente a SUT e TAS in quanto uno è un componente dell'ambiente dell'altro). Ogni cambiamento applicato ad un SUT o ad una TAS può impattare l'altro e quindi la gestione di questi cambiamenti deve comprendere sia il SUT che la TAS.

Due approcci alla sincronizzazione fra i processi di sviluppo del SUT e della TAS sono descritti nelle Figure 3 e 4.

La Figura 3 mostra un approccio in cui i due processi di SDLC per il SUT e la TAS sono principalmente sincronizzati in due fasi: (1) l'analisi della TAS è basata sulla progettazione del SUT, che a sua volta è basata sull'analisi del SUT e (2) il testing del SUT fa uso della TAS rilasciata.

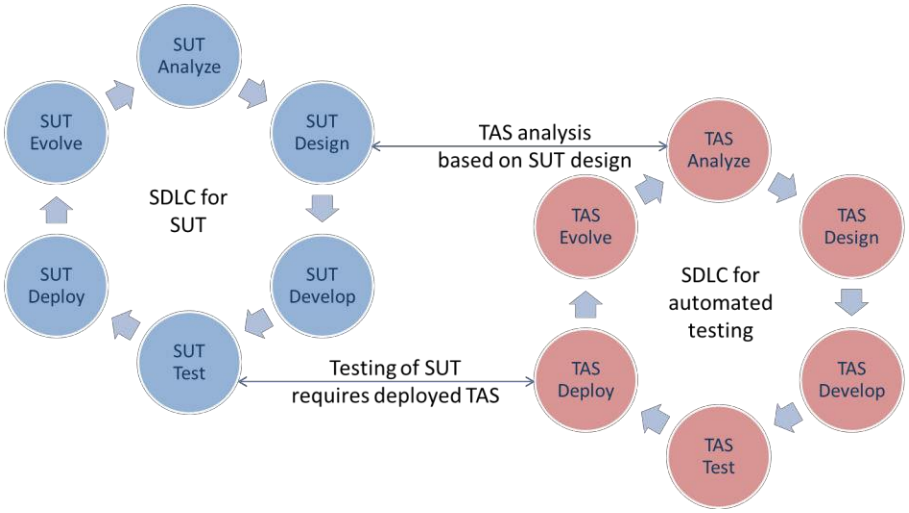

Figura 3: Esempio 1 della sincronizzazione dei processi di sviluppo di TAS e SUT

La Figura 4 mostra un approccio ibrido con sia il testing manuale che automatizzato. Ogni volta che i test manuali sono usati prima che i test siano automatizzati o ogni volta che i test manuali e automatizzati sono usati insieme, l'analisi della TAS deve essere basata sia sulla progettazione del SUT che sui test manuali. In questo modo, la TAS è sincronizzata con entrambi. Il secondo punto più importante di sincronizzazione per tale approccio è lo stesso del precedente: il testing del SUT richiede che siano stati rilasciati i test, che nel caso di test manuali potrebbe consistere solo nell'eseguire le procedure dei test manuali.

© International Software Testing Qualifications Board

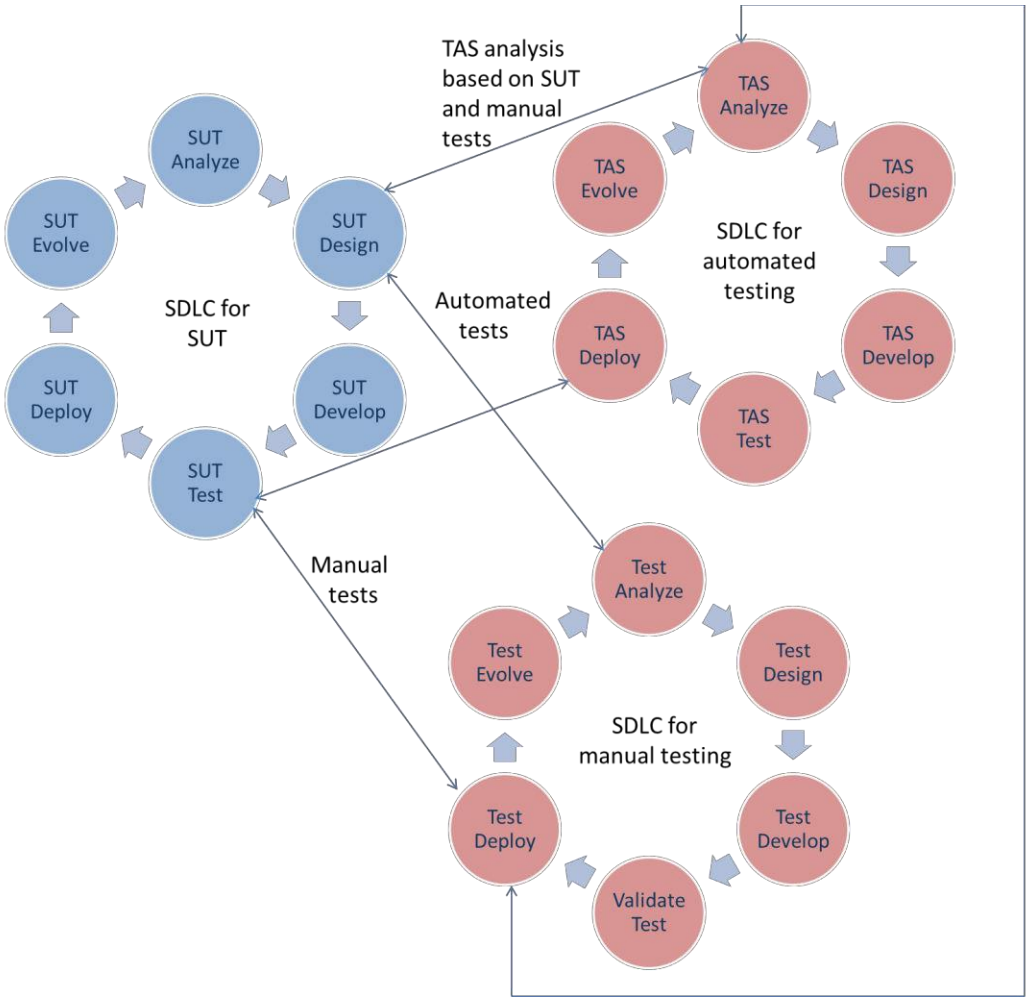

Figura 4: Esempio 2 della sincronizzazione dei processi di sviluppo di TAS e SUT

3.3.4 Realizzare il Riuso nella TAS

Per riuso di una TAS si intende il riuso degli artefatti della TAS (da ogni livello della sua architettura) attraverso linee di prodotto, framework di prodotto, domini di prodotto e/o famiglie di progetti. I requisiti per il riuso derivano dalla rilevanza degli artefatti della TAS per le altre varianti del prodotto, prodotti e/o progetti. Artefatti riusabili di una TAS possono includere:

- (Parti di) modelli di test di obiettivi di test, scenari di test, componenti di test o dati di test
- (Parti di) casi di test, dati di test, procedure di test o le stesse librerie di test
- Il motore del test e/o il framework dei report di test
- Gli adattatori ai componenti e/o alle interfacce del SUT

Mentre gli aspetti di riuso sono già stabiliti quando la TAA è definita, la TAS può aiutare ad incrementare la capacità di riuso:

- Seguendo la TAA o rivedendola e aggiornandola quando necessario

© International Software Testing Qualifications Board

- Documentando gli artefatti della TAS in modo tale che siano compresi facilmente e possano essere incorporati in nuovi contesti
- Assicurando la correttezza di ogni artefatto della TAS in modo tale che l'uso in nuovi contesti sia supportato dalla sua alta qualità

È importante notare che mentre progettare per il riuso riguarda principalmente la TAA, la manutenzione e il miglioramento del riuso rappresentano un impegno in tutto il ciclo di vita della TAS. Sono richiesti continui ragionamenti e sforzi per rendere il riuso concreto, per misurare e dimostrare il valore aggiunto del riuso e per evangelizzare gli altri a riusare le TAS esistenti.

3.3.5 Supporto per una varietà di sistemi target

Per supporto della TAS per una varietà di sistemi target si intende la capacità di una TAS di testare configurazioni differenti di un prodotto software. Per configurazioni differenti si intende quanto segue:

- Numero e interconnessioni dei componenti del SUT
- Ambienti (sia software che hardware) su cui girano i componenti del SUT
- Tecnologie, linguaggi di programmazione o sistemi operativi usati per implementare i componenti del SUT
- Librerie e pacchetti in uso da parte dei componenti del SUT
- Strumenti usati per realizzare i componenti del SUT

Mentre i primi quattro aspetti impattano la TAS ad ogni livello di test, l'ultimo si applica principalmente ai livelli di testing di componente e di integrazione.

La capacità di una TAS di testare differenti configurazioni di un prodotto software è stabilita quando è definita la TAA. Comunque, la TAS deve implementare la capacità di gestire le variazioni tecniche e deve rendere possibile la gestione delle funzionalità e dei componenti della TAS necessari alle differenti configurazioni di un prodotto software.

La gestione della diversità della TAS in relazione alla diversità del prodotto software può essere affrontata in modi differenti:

- La gestione della versione/ configurazione della TAS e del SUT può essere usata per fornire le rispettive versioni e configurazioni della TAS e del SUT corrispondenti
- La parametrizzazione della TAS può essere usata per adattare una TAS ad una configurazione del SUT

È importante notare che, mentre progettare per la variabilità della TAS riguarda principalmente la TAA, la manutenzione e il miglioramento per la variabilità rappresentano un impegno in tutto il ciclo di vita della TAS. Sono richiesti continui ragionamenti e sforzi per rivedere, aggiungere e anche rimuovere opzioni e forme di variabilità.

© International Software Testing Qualifications Board

4. Rischi di Deployment e Contingenze - 150 min.

Termini
rischio, mitigazione del rischio, valutazione del rischio, rischio di prodotto

Obiettivi di apprendimento su Rischi di Deployment e Contingenze

4.1 Selezione di un Approccio di Automazione del Testing e Pianificazione del Deployment/Rollout.

ALTA-E-4.1.1 (K3) Applicare le linee guida a supporto di efficaci attività di pilota e deployment di uno strumento di test

4.2 Valutazione del Rischio e Strategie di Mitigazione

ALTA-E-4.2.1 (K4) Analizzare i rischi di deployment , identificare i problemi tecnici che potrebbero portare al fallimento di un progetto di automazione del testing e prevedere le strategie di mitigazione

4.3 Manutenzione dell'Automazione del Testing

ALTA-E-4.3.1 (K2) Comprendere quali fattori supportano e influenzano la manutenibilità di una TAS

© International Software Testing Qualifications Board

4.1 Selezione di un Approccio di Automazione del Testing e Pianificazione del Deployment/Rollout

Ci sono due attività principali coinvolte nell'implementazione e nel rollout di una TAS: pilota e deployment. I passi che costituiscono queste due attività varieranno a seconda del tipo di TAS e della situazione specifica.

Per il pilota, devono essere considerati almeno i seguenti passi:

- Identificare un progetto adatto
- Pianificare il pilota
- Condurre il pilota
- Valutare il pilota

Per il deployment, devono essere considerati almeno i seguenti passi:

- Identificare il/ progetto/i da cui iniziare
- Deploy della TAS sui progetti selezionati
- Monitorare e valutare la TAS nei progetti dopo un periodo predefinito
- Effettuare il rollout al resto degli altri progetti/organizzazione

4.1.1 Progetto Pilota

L'implementazione di uno strumento di solito parte con un progetto pilota. L'obiettivo del progetto pilota è assicurare che la TAS possa essere usata per ottenere i vantaggi pianificati. Gli obiettivi del progetto pilota includono:

- Imparare maggiori dettagli sulla TAS.
- Vedere come la TAS si adegua a processi, procedure e strumenti esistenti; identificare come potrebbe essere necessario cambiarli. (È di solito preferibile modificare la TAS in modo che si adatti ai processi/procedure esistenti. Se questi devono essere modificati per "supportare la TAS", ciò dovrebbe almeno risultare in un miglioramento dei processi stessi).
- Progettare le interfacce dell'automazione per soddisfare le necessità dei tester.
- Decidere le modalità standard per usare, gestire, conservare e manutenere la TAS e gli altri oggetti di test inclusa l'integrazione con la gestione della configurazione e la gestione delle modifiche (per es. decidere la convenzioni dei nomi per file e test, creare librerie e definire la modularità delle suite di test).
- Identificare le metriche e i metodi di misura per monitorare l'uso dell'automazione del testing, incluse usabilità, manutenibilità ed espandibilità.
- Valutare se i benefici possono essere raggiunti a costi ragionevoli. Questo darà la possibilità di ridurre le aspettative una volta che le TAS siano state usate.
- Determinare quali competenze sono richieste e quali di esse sono disponibili e quali invece mancano.

Identificare un progetto adatto
Il progetto pilota deve essere selezionato con cura usando le seguenti linee guida:

- Non selezionare un progetto critico. Quando il deployment della TAS causa ritardi, ciò non deve avere impatti importanti su progetti critici. Il deployment della TAS costerà tempo all'inizio. Il team di progetto ne deve essere consapevole.

© International Software Testing Qualifications Board

- Non selezionare un progetto banale. Un progetto banale non è un buon candidato poiché il successo del deployment dell'automazione non implica il successo su progetti non banali, e quindi aggiunge poco alle informazioni necessarie per il deployment.
- Coinvolgere le persone necessarie (incluso il management) nel processo di selezione.
- Il SUT del progetto pilota dovrebbe essere un buon punto di riferimento per gli altri progetti all'interno della struttura organizzativa, per es. il SUT dovrebbe contenere componenti GUI rappresentativi di quelli che dovranno essere automatizzati.

Pianificare il pilota
Il pilota dovrebbe essere trattato come un normale progetto di sviluppo: fare un piano, riservare budget e risorse, effettuare report degli avanzamenti, definire milestone, ecc. Un ulteriore punto di attenzione è assicurarsi che le persone che lavoreranno sull'uso della TAS possano spendere abbastanza sforzo anche quando altri progetti richiederanno risorse per le loro attività. È importante avere l'appoggio del management, in modo particolare sulle risorse condivise. Queste persone probabilmente non potranno lavorare a tempo pieno sul pilota.

Quando la TAS non è stata fornita da un venditore, ma è sviluppata in casa, gli sviluppatori di tale TAS dovranno essere coinvolti nelle attività di deployment.

Condurre il pilota
Effettuare il pilota del deployment e porre attenzione ai punti seguenti:

- La TAS fornisce le funzionalità attese (e promesse dal venditore)? Se no, è necessario che quanto richiesto sia fornito appena possibile. Quando la TAS è sviluppata in casa i suoi sviluppatori devono dare assistenza al deployment, fornendo le funzionalità mancanti.
- La TAS e il processo preesistente si supporto reciprocamente? Se no, devono essere allineati.

Valutare il pilota
Far partecipare tutti gli interessati alla valutazione.

4.1.2 Deployment

Una volta che il pilota è stato valutato, il deployment della TAS al resto dell'organizzazione deve essere effettuato solo se il pilota si è concluso con successo. Il rollout deve avvenire in modo incrementale ed essere ben gestito. Fattori di successo per il deployment includono:

- Un rollout incrementale: effettuare il rollout al resto dell'organizzazione a passi, per incrementi successivi. In questo modo, il supporto ai nuovi utenti avviene a "ondate" piuttosto che tutto in una volta. Ciò consente all'uso della TAS di crescere passo per passo. Eventuali colli di bottiglia possono essere identificati e risolti prima che divengano problemi reali. Ulteriori licenze possono essere aggiunte solo quando necessarie.
- Adattare e migliorare i processi per adattarli all'uso della TAS: quando utenti differenti usano la TAS, vengono impattati dalla TAS processi differenti che hanno bisogno di essere sincronizzati con la TAS, o la TAS stessa può aver bisogno di (piccole) modifiche per adattarsi ai processi.
- Fornire formazione e affiancamento/consulenza ai nuovi utenti: i nuovi utenti hanno bisogno di formazione e affiancamento nell'uso di una nuova TAS. Ci si deve assicurare che ciò venga effettuato. Sessioni di formazione/workshop devono essere fornite agli utenti prima che usino realmente la TAS.

© International Software Testing Qualifications Board

- Definire linee guida per l'uso: è possibile scrivere linee guida, checklist e FAQ per l'uso della TAS. Ciò può prevenire il moltiplicarsi di richieste di supporto.
- Implementare un modo per raccogliere informazioni sull'uso corrente: ci deve essere una modalità automatica per raccogliere informazioni sull'uso della TAS. Idealmente non solo l'uso in sé, ma anche quali parti della TAS (quali funzionalità) si stanno utilizzando. In questo modo, l'uso della TAS può essere facilmente monitorato.
- Monitorare uso, benefici e costi della TAS: monitorare l'uso della TAS per un certo periodo di tempo indica se la TAS è usata realmente. Questa informazione può anche essere usata per ricalcolare il business case (per es. quanto tempo è stato risparmiato, quanti problemi prevenuti).
- Fornire supporto per i team di test e di sviluppo per una data TAS.
- Raccogliere le "lesson learned" da tutti i team: svolgere riunioni di valutazione/retrospettiva con i differenti team che usano la TAS. In questo modo, possono essere identificate delle "lesson learned". I team sentiranno che il loro contributo è necessario e richiesto per migliorare l'uso della TAS.
- Identificare e implementare i miglioramenti: in base ai feedback del team e al monitoraggio della TAS, identificare e implementare dei passi di miglioramento. Inoltre, comunicare chiaramente quanto fatto a tutte le persone coinvolte.

4.1.3 Deployment della TAS all'interno del Ciclo di Vita del Software

Il deployment di una TAS dipende molto dalla fase di sviluppo del progetto software che sarà testato dalla TAS.

Di solito, una nuova TAS o una sua nuova versione viene rilasciata o all'inizio del progetto o quando si raggiunte una milestone, come un consolidamento del codice o la fine di uno sprint. Questo perché le attività di deployment, con tutti i test e le modifiche necessarie, richiedono tempo e lavoro. Questo è anche un buon modo per mitigare il rischio che la TAS non funzioni e causi interruzioni nel processo di automazione del testing. Comunque, se ci sono anomalie critiche che devono essere risolte per la TAS o se un componente dell'ambiente in cui essa viene eseguita deve essere cambiato, allora il deployment sarà fatto indipendentemente dalla fase di sviluppo del SUT.

4.2 Valutazione del Rischio e Strategie di Mitigazione

Problemi tecnici possono portare a rischi di prodotto o di progetto. Problemi tecnici tipici comprendono:

- Troppa astrazione può creare difficoltà nel comprendere cosa accade realmente (per es. con le keyword)
- Data-driven: le tabelle dei dati possono diventare troppo grandi/complesse /pesanti
- Dipendenza della TAS dall'uso di alcune librerie del sistema operativo o altri componenti che possono non essere disponibili in tutti gli ambienti target del SUT

Rischi di progetto tipici possono includere:

- Problemi di staffing: ottenere le persone giuste per manutenere il codice può essere difficoltoso
- Nuovi rilasci del SUT possono far sì che la TAS non operi in modo corretto
- Ritardi nell'introdurre l'automazione
- Ritardi nell'aggiornamento della TAS in base ai cambiamenti apportati al SUT
- La TAS può non catturare gli oggetti (non standard) che si desidera controllare

© International Software Testing Qualifications Board

Potenziali punti di fallimento per il progetto della TAS possono includere:

- Migrazione ad un ambiente differente
- Deployment nell'ambiente target
- Nuovo rilascio dallo sviluppo

Ci sono molteplici strategie di mitigazione del rischio che possono essere impiegate per affrontare tali aree di rischio. Tali strategie saranno discusse di seguito.

La TAS ha un suo proprio ciclo di vita del software, sia in caso di sviluppo in casa che per soluzioni acquistate dall'esterno. Una cosa da ricordare è che la TAS, come ogni altro software, ha bisogno di controllo delle versioni e funzionalità documentate. Altrimenti, diventa molto difficile installare parti separate della TAS e farle funzionare insieme o in determinati ambienti.

Inoltre, ci deve essere una procedura di installazione documentata, chiara e facile da seguire. Questa procedura è dipendente dalla versione; quindi deve essere anch'essa inclusa sotto il controllo di versione.

Ci sono due distinti casi quando si avvia una TAS:

1. Deployment iniziale
2. Deployment per manutenzione: la TAS già esiste e deve essere manutenuta

Prima di iniziare con il primo rilascio di una TAS, è importante essere sicuri che possa girare nel suo proprio ambiente, sia isolata da cambiamenti casuali e i casi di test possano essere aggiornati e gestiti. Sia la TAS che la sua infrastruttura devono essere manutenuti.

Nel caso del primo deployment, sono necessari i seguenti passi base:

- Definire l'infrastruttura in cui la TAS opererà
- Creare l'infrastruttura per la TAS
- Creare una procedura per manutenere la TAS e la sua infrastruttura
- Creare una procedura per manutenere le suite di test che la TAS eseguirà

I rischi legati al primo rilascio includono:

- Il tempo totale di esecuzione della suite di test può essere più lungo del tempo di esecuzione pianificato per il ciclo di test. In questo caso è importante essere sicuri che la suite di test abbia il tempo sufficiente per essere eseguita interamente prima che inizi il successivo ciclo di test pianificato.
- Esistono problemi di installazione e configurazione (per es. setup e caricamento iniziale del database, start/stop dei servizi). In generale, la TAS ha bisogno di un modo efficace per settare le precondizioni necessarie per i casi di test automatizzati all'interno dell'ambiente di test.

Per i deployment per manutenzione, ci sono considerazioni aggiuntive. La TAS stessa ha bisogno di evolvere e i suoi aggiornamenti devono essere rilasciati in produzione. Prima di rilasciare una versione aggiornata di una TAS in produzione, è necessario che sia testata come ogni altro software. È quindi necessario controllare le nuove funzionalità, per verificare che la suite di test possa essere eseguita dalla TAS aggiornata, che i report possano essere inviati e che non ci siano problemi di prestazioni o altre regressioni funzionali. In alcuni casi può essere necessario cambiare l'intera suite di test per adattarla alla nuova versione della TAS.

Quando accadono problemi in deployment per manutenzione, sono necessari i seguenti passi:

© International Software Testing Qualifications Board

- Fare una valutazione dei cambiamenti della nuova versione della TAS rispetto alla precedente
- Testare la TAS per le nuove funzionalità e per la regressione
- Controllare se la suite di test necessiti di essere adattata alla nuova versione della TAS

Un aggiornamento provoca anche i seguenti rischi e le corrispondenti azioni di mitigazione:

- La suite di test deve cambiare per essere eseguita sulla TAS aggiornata: fare i cambiamenti necessari alla suite di test e testarli prima di rilasciarli sulla TAS.
- Stub, driver e interfacce usati nel testing devono essere cambiati per adattarli alla TAS aggiornata: fare i cambiamenti necessari alla test harness e testarla prima di usarla sulla TAS
- L'infrastruttura deve essere cambiata per adattarsi alla TAS aggiornata: fare una valutazione dei componenti dell'infrastruttura che devono essere cambiati, effettuare i cambiamenti e testarli con la TAS aggiornata.
- La TAS aggiornata ha difetti aggiuntivi o problemi prestazionali: svolgere un'analisi dei rischi rispetto ai benefici. Se i problemi aperti rendono impossibile aggiornare la TAS, può essere meglio non procedere con l'aggiornamento o aspettare la prossima versione della TAS. Se i problemi sono trascurabili rispetto ai benefici, la TAS può ancora essere aggiornata. Si deve creare una release note con i problemi aperti per avvisare i TAE e le altre persone coinvolte e cercare di avere una stima su quando i problemi potranno essere risolti.

4.3 Manutenzione dell'Automazione del Testing

Sviluppare soluzioni di automazione dei testing non è banale. È necessario che siano modulari, scalabili, comprensibili, affidabili e testabili. Per aggiungere ulteriore complessità, le soluzioni di automazione del testing – come ogni altro sistema software – devono evolvere. A seguito di cambiamenti interni o per cambiamenti nell'ambiente in cui devono operare, la manutenzione è un importante aspetto della progettazione di una TAS. Manutenere la TAS adattandola a nuovi tipi di sistemi da testare, arricchendone il supporto per nuovi ambienti software o rendendola conforme a nuove leggi o regolamenti, aiuta ad assicurare un'operatività affidabile e sicura. Ciò ottimizza anche il tempo di vita e le prestazioni della TAS.

4.3.1 Tipi di Manutenzione

La manutenzione è fatta su TAS esistenti e in uso ed è innescata da modifiche, migrazioni o ritiro di sistemi. Questo processo può essere strutturato nelle seguenti categorie:

- Manutenzione preventiva – I cambiamenti sono fatti per far supportare alla TAS un numero maggiore di tipi di test, test su molteplici interfacce, test su molteplici versioni del SUT o supportare l'automazione del testing per un nuovo SUT.
- Manutenzione correttiva – I cambiamenti sono fatti per correggere anomalie della TAS. Il miglior modo per manutenere una TAS operativa, riducendo così il rischio nell'usarla, è per mezzo dell'esecuzione di regolari test di manutenzione.
- Manutenzione di perfezionamento – La TAS è ottimizzata e sono risolti problemi non funzionali. Può coinvolgere prestazioni, usabilità, robustezza e affidabilità della TAS.
- Manutenzione adattativa – Quando nuovi sistemi software sono lanciati sul mercato (sistemi operativi, gestori di database, web browser, ecc.) può essere richiesto che la TAS li supporti. Inoltre può accadere che la TAS debba essere resa conforme a nuove leggi, regolamenti o requisiti specifici del settore. In questo caso, sono fatti cambiamenti alla TAS per adeguarla

© International Software Testing Qualifications Board

secondo tali esigenze. Da notare: di solito, la conformità a leggi e regolamenti provoca una manutenzione obbligatoria con specifiche regole, requisiti e alcune volte requisiti di auditing. Inoltre, quando gli strumenti di integrazione sono aggiornati e nuove versioni sono create, gli endpoint degli strumenti di integrazione devono essere manutenuti per garantirne il funzionamento.

4.3.2 Ambito e Approccio

La manutenzione è un processo che può toccare tutti gli strati e i componenti di una TAS. La sua portata dipende da:

- La dimensione e la complessità della TAS
- La dimensione del cambiamento
- Il rischio del cambiamento

Dato che la manutenzione riguarda una TAS operativa, è necessaria un'analisi di impatto per determinare come il sistema possa essere influenzato dai cambiamenti. A seconda dell'impatto, i cambiamenti devono essere introdotti in modo incrementale e devono essere effettuati dei test dopo ciascun passo per assicurare la continuità di funzionamento della TAS. Nota: manutenere la TAS può essere difficile se le sue specifiche e la documentazione non sono aggiornate.

Poiché l'efficienza temporale è il principale fattore che contribuisce al successo dell'automazione del testing, diventa critico avere delle buone pratiche per la manutenzione della TAS che includono:

- Le procedure di deployment ed uso della TAS devono essere chiare e documentate
- Le dipendenze da terze parti devono essere documentate, insieme con gli svantaggi e i problemi conosciuti
- La TAS deve essere modulare, così che le sue parti possano essere facilmente sostituite
- La TAS deve essere eseguita in un ambiente che sia sostituibile o con componenti sostituibili
- La TAS deve separare gli script di test dal TAF stesso
- La TAS deve girare isolata dall'ambiente di sviluppo, cosicché i cambiamenti alla TAS non influenzeranno negativamente l'ambiente di test
- La TAS insieme all'ambiente, la suite di test e gli artefatti del testware devono essere sotto gestione della configurazione

Ci sono anche considerazioni, come le seguenti, sulla manutenzione di componenti di terze parti e altre librerie:

- Molto spesso accade che la TAS userà componenti di terze parti per eseguire i test. Può anche essere il caso che la TAS dipenda da librerie prodotte da terze parti (per es. librerie di automazione della UI). Tutte le componenti di terze parti della TAS devono essere documentate e sotto gestione della configurazione.
- È necessario avere un piano nel caso in cui queste componenti esterne debbano essere modificate o corrette. La persona responsabile della manutenzione della TAS deve sapere chi contattare o dove sottoporre un problema.
- Ci deve essere la documentazione riguardante la licenza sotto la quale le componenti di terze parti sono rilasciate, cosi che siano disponibili le informazioni sulla possibilità di modificarle, a quale livello e da parte di chi.

© International Software Testing Qualifications Board

- Per ognuna delle componenti di terze parti, è necessario ottenere informazioni sugli aggiornamenti e le nuove versioni. Manutenere tali componenti e librerie aggiornate è un'azione preventiva che ripaga l'investimento nel lungo termine.

Considerazioni per standard di codifica dei nomi e altre convenzioni includono:

- L'idea di standard di codifica dei nomi e altre convenzioni ha una semplice ragione: la suite di test e la TAS stessa devono essere facili da leggere, da capire, modificare e manutenere. Questo consente di risparmiare tempo nel processo di manutenzione e minimizza anche il rischio di introdurre regressioni o correzioni sbagliate che potrebbero altrimenti non essere facilmente evitate.
- È più facile introdurre nuove persone nel progetto di automazione del testing quando si usano convenzioni standard di codifica dei nomi.
- Gli standard di codifica dei nomi possono riguardare variabili e file, scenari di test, keyword e parametri di keyword. Altre convenzioni riguardano i prerequisiti e le post-azioni per l'esecuzione dei test, il contenuto dei dati di test, l'ambiente di test, lo stato dell'esecuzione dei test e i log e i report delle esecuzioni.
- Tutti gli standard e le convenzioni devono essere concordati e documentati quando si inizia un progetto di automazione del testing.

Considerazioni sulla documentazione includono:

- La necessità di una documentazione buona e aggiornata sia per gli scenari di test che per la TAS è abbastanza ovvia, ma ci sono due questioni connesse: qualcuno la deve scrivere e qualcuno la deve aggiornare.
- Mentre il codice di uno strumento di test può essere o auto-documentante o documentato in modo semiautomatico, tutto il progetto, le componenti, le integrazioni con gli altri sistemi, le dipendenze e le procedure di installazione devono essere documentate da qualcuno.
- È una buona pratica introdurre la scrittura della documentazione come parte del processo di sviluppo. Un'attività non dovrebbe essere considerata conclusa a meno che sia documentata o la relativa documentazione non sia aggiornata.

Considerazioni sul materiale per la formazione includono:

- Se la documentazione della TAS è ben scritta, può essere usata come base del materiale per la formazione sulla TAS.
- Il materiale per la formazione è una combinazione di specifiche funzionali della TAS, progettazione e architettura della TAS, deployment e manutenzione della TAS, uso della TAS (manuale utente), esempi pratici ed esercizi, trucchi e suggerimenti.
- La manutenzione del materiale per la formazione consiste nella scrittura iniziale e nella sua revisione periodica. È svolta in pratica da membri del team incaricati come formatori sulla TAS e di solito avviene verso la fine di un'iterazione del ciclo di vita del SUT (alla fine di sprint, per esempio).

© International Software Testing Qualifications Board

5. Report e Metriche per l'Automazione del Testing - 165 min.

Termini
Densità dei difetti nel codice dell'automazione, copertura, matrice di tracciabilità, sforzo di test manuale equivalente (equivalent manual test effort), metriche, logging del test, reporting del test

Obiettivi di apprendimento su Report e Metriche per l'Automazione del Testing

5.1 Selezione di Metriche per la TAS
ALTA-E-5.1.1 (K2) Classificare le metriche che possono essere usate per monitorare la strategia dell'automazione del testing e la sua efficacia

5.2 Implementazione delle Misure
ALTA-E-5.2.1 (K3) Implementare metodi di raccolta di metriche per supportare requisiti tecnici e di gestione. Spiegare come le misure dell'automazione del testing possono essere implementate

5.3 Logging della TAS e del SUT
ALTA-E-5.3.1 (K4) Analizzare il logging dei test sia dei dati del SUT che della TAS

5.4 Reportistica dell'Automazione del Testing
ALTA-E-5.4.1 (K2) Spiegare come un report dell'esecuzione dei test è costruito e pubblicato

© International Software Testing Qualifications Board

5.1 Selezione di Metriche per la TAS

Questo paragrafo tratta le metriche che possono essere usate per monitorare la strategia di automazione del testing e l'efficacia ed efficienza della TAS. Queste sono separate dalle metriche legate al SUT, usate per monitorare il SUT e il testing del SUT (funzionale e non funzionale), che sono selezionate dal Test Manager per tutto il progetto. Le metriche dell'automazione del testing consentono al TAM e al TAE di tracciare gli avanzamenti rispetto agli obiettivi dell'automazione del testing e monitorare l'impatto dei cambiamenti apportati alla soluzione di automazione del testing.

Le metriche della TAS possono essere divise in due gruppi: esterne ed interne. Le metriche esterne sono quelle usate per misurare l'impatto della TAS sulle altre attività (in particolare le attività di testing). Le metriche interne sono quelle usate per misurare efficacia ed efficienza della TAS nel raggiungere i suoi obiettivi.

Le metriche della TAS tipicamente includono le seguenti:

- Metriche esterne della TAS
  - Benefici dell'automazione
  - Sforzo per realizzare i test automatizzati
  - Sforzo per analizzare gli incidenti nei test automatizzati
  - Sforzo per manutenere i test automatizzati
  - Rapporto tra fallimenti (esiti negativi) e difetti
  - Tempo per eseguire i test automatizzati
  - Numero di casi di test automatizzati
  - Numero di risultati passati e falliti
  - Numero di risultati di falsi fallimenti (falsi esiti negativi) e falsi passati
  - Copertura del codice
- Metriche interne della TAS
  - Metriche di scripting dello strumento
  - Densità dei difetti nel codice dell'automazione
  - Velocità ed efficienza dei componenti della TAS

Queste metriche sono descritte di seguito.

Benefici dell'automazione
È di particolare importanza misurare e produrre dei report dei benefici di una TAS. Questo perché i costi (in termini di numero di persone coinvolte in un dato periodo di tempo) sono facili da vedere. Le persone che lavorano fuori dell'ambito del testing saranno capaci di farsi un'impressione dei maggiori costi, ma possono non vedere i benefici ottenuti.

Ogni misura dei benefici dipenderà dall'obiettivo della TAS. Tipicamente questo può essere un risparmio di tempo o di sforzo, un incremento nel numero di test eseguiti (estensione o profondità di copertura o frequenza di esecuzione) o qualche altro vantaggio come un aumento della ripetibilità, un più grande uso delle risorse o un numero più basso di errori manuali. Possibili tipi di misure includono:

- Numero di ore risparmiate di sforzo per il testing manuale
- Riduzione del tempo per eseguire il testing di regressione
- Numero di cicli aggiuntivi di esecuzione dei test ottenuto

© International Software Testing Qualifications Board

- Numero o percentuale di test aggiuntivi eseguiti
- Percentuale di casi di test automatizzati rispetto all'intero insieme dei casi di test (sebbene i casi di test automatizzati non possano essere facilmente comparati con quelli manuali)
- Incremento nella copertura (requisiti, funzionalità, strutturale)
- Numero di difetti rilevati in anticipo grazie alla TAS (quando il beneficio medio dei difetti trovati in anticipo è conosciuto, questo beneficio può essere calcolato in termini di somma di costi evitati)
- Numero di difetti rilevati grazie alla TAS che non sarebbero stati rilevati dal testing manuale (per es. difetti di affidabilità)

È da notare che generalmente l'automazione del testing risparmia sforzo di testing manuale. Questo sforzo può essere dedicato ad altri tipi di testing (manuale) (per es. testing esplorativo). I difetti trovati da questi test aggiuntivi possono essere visti come benefici indiretti della TAS, poiché l'automazione del testing rende possibile l'esecuzione di questi test manuali. Senza la TAS questi test non sarebbero stati eseguiti e di conseguenza i difetti in più non sarebbero stati trovati.

Sforzo per realizzare i test automatizzati
Lo sforzo per automatizzare i test è uno dei costi chiave associati all'automazione del testing. Esso è spesso maggiore del costo di eseguire gli stessi test manualmente e quindi può essere un impedimento all'aumento dell'uso dell'automazione del testing. Mentre il costo per implementare uno specifico test automatizzato dipenderà in gran parte dal test stesso, avranno un impatto anche altri fattori come l'approccio di scripting usato, la familiarità con lo strumento di test, l'ambiente e il livello di competenza del test automation engineer.

Poiché test più grandi o complessi tipicamente richiederanno tempi più lunghi per essere automatizzati rispetto a test più brevi o semplici, il calcolo del costo di realizzazione per l'automazione del testing può essere basato su un tempo di realizzazione medio. Questo può essere ulteriormente rifinito considerando il costo medio per uno specifico insieme di test come quelli che riguardano la stessa funzione o quelli ad un dato livello di test. Un altro approccio è esprimere il costo di realizzazione come un fattore dello sforzo richiesto per eseguire il test manualmente (equivalent manual test effort, EMTE). Per esempio, può essere che sia necessario due volte lo sforzo per il test manuale per automatizzare un caso di test, o due volte l'EMTE.

Sforzo per analizzare gli incidenti nei test automatizzati
Analizzare gli esiti negativi dei test sul SUT scoperti tramite l'esecuzione del testing automatizzato può essere significativamente più complesso che per un test eseguito manualmente perché gli eventi che portano al fallimento di un test manuale sono spesso conosciuti dal tester che esegue il test. Ciò può essere mitigato come descritto nel livello di progettazione nel paragrafo 3.1.4 e nel livello di reporting nei paragrafi 5.3 e 5.4. Questa misura può essere espressa come una media per casi di test falliti o può essere espressa come fattore dell'EMTE. Quest'ultima è particolarmente adatta quando i test automatizzati variano significativamente in complessità e lunghezza di esecuzione.

I logging disponibili del SUT e della TAS svolgono un ruolo cruciale nell'analisi degli esiti negativi. Il logging deve fornire informazioni sufficienti per eseguire tali analisi in modo efficace. Caratteristiche importanti del logging includono:

- I logging del SUT e della TAS devono essere sincronizzati
- La TAS deve registrare il comportamento atteso e attuale
- La TAS deve registrare le azioni da eseguire.

© International Software Testing Qualifications Board

Il SUT, d'altro canto, dovrebbe effettuare il logging di tutte le azioni che sono eseguite (senza riguardo al fatto che l'azione sia il risultato di testing manuale o automatizzato). Ogni errore interno dovrebbe essere registrato e dovrebbero essere disponibili tutti i crash dump e gli stack trace.

Sforzo per manutenere i test automatizzati
Lo sforzo richiesto per manutenere i test automatizzati sincronizzati con il SUT può essere molto significativo e alla fine può superare i benefici ottenuti con la TAS. Questa è stata la causa del fallimento per molti sforzi di automazione. Monitorare lo sforzo di manutenzione è perciò importante per evidenziare quando devono essere intrapresi degli interventi per ridurre lo sforzo di manutenzione o almeno prevenire che cresca in modo incontrollato.

Misure dello sforzo di manutenzione possono essere espresse come totale di tutti i test automatizzati che richiedono manutenzione per ogni release del SUT. Possono anche essere espressi con una media per test automatizzato aggiornato o come fattore dell'EMTE.

Una metrica correlata è il numero o la percentuale dei test che richiedono lavoro di manutenzione.

Quando lo sforzo di manutenzione per i test automatizzati è conosciuto (o può essere derivato), tale informazione può svolgere un ruolo cruciale nel decidere se o implementare o meno certe funzionalità o correggere certi difetti. Lo sforzo richiesto per manutenere i casi di test dovuto al cambiamento del software deve essere considerato insieme al cambiamento del SUT.

Rapporto tra fallimenti (esiti negativi) e difetti
Un problema comune con i test automatizzati è che molti di loro possono fallire per la stessa ragione: un singolo difetto nel software. Mentre lo scopo dei test è evidenziare i difetti nel software, avere più di un test che rileva lo stesso difetto è inefficiente. Questo è vero in particolare nel caso dei test automatizzati poiché lo sforzo richiesto per analizzare ciascun test fallito può essere significativo. Misurare il numero di test automatizzati che falliscono per un dato difetto può aiutare ad indicare dove ciò può essere un problema. La soluzione risiede nella progettazione dei test automatizzati e nella loro selezione per l'esecuzione.

Tempo per eseguire i test automatizzati
Una delle metriche più facili da determinare è il tempo che occorre per eseguire i test automatizzati. All'inizio della TAS ciò può non essere importante, ma all'aumentare dei casi di test automatizzati, tale metrica può diventare piuttosto importante.

Numero di casi di test automatizzati
Questa metrica può essere usata per mostrare l'avanzamento fatto dal progetto di automazione del testing. Tuttavia si deve tenere contro che il solo numero di casi di test automatizzati non fornisce molte informazioni; per esempio, non indica che la copertura dei test è aumentata.

Numero di risultati passati e falliti
Questa è una metrica comune e traccia quanti test automatizzati sono passati e quanti falliti nell'ottenere il risultato atteso. I fallimenti devono essere analizzati per determinare se siano dovuti ad un difetto nel SUT o a problemi esterni come problemi di ambiente o della TAS stessa.

© International Software Testing Qualifications Board

Numero di risultati di falsi fallimenti (falsi esiti negativi) e falsi passati
Come visto in diverse metriche precedenti, può essere necessario diverso tempo per analizzare i test falliti. Questo è anche più frustrante quando si scopre che si tratta di un falso allarme. Ciò avviene quando il problema è nella TAS o nel caso di test, ma non nel SUT. È importante che il numero di falsi allarmi (e lo sforzo potenzialmente sprecato) siano mantenuti bassi. Falsi fallimenti possono ridurre la confidenza nella TAS. Al contrario, risultati falsi passati possono essere più pericolosi. Quando si verifica un falso passato, c'è stato un esito negativo nel SUT che non è stato identificato dall'automazione del testing e quindi viene riportato un superamento del test. In questo caso, un difetto potenziale può sfuggire al rilevamento. Questo può accadere perché la verifica di un risultato non era stata fatta correttamente, era stato usato un oracolo del test non valido o il caso di test si attendeva un risultato errato.

Si noti che falsi allarmi possono essere causati da difetti nel codice del test (si veda la metrica "Densità dei difetti nel codice dell'automazione"), ma può anche essere causato da un SUT instabile che si sta comportando in modo non prevedibile (per es. un time out). I test hook possono anche causare falsi allarmi dovuti al livello di intrusione che causano.

Copertura del codice
Conoscere la copertura del codice del SUT fornita dai differenti casi di test può rivelare utili informazioni. La copertura può essere misurata anche ad alto livello, per es. la copertura del codice della suite di test di regressione. Non c'è una percentuale assoluta che indica una copertura adeguata e la copertura del 100% del codice non è ottenibile in nessuna applicazione software se non in quelle più semplici. Comunque, è opinione generale che sia meglio avere una maggiore copertura per ridurre il rischio complessivo del rilascio del software. Questa metrica può anche indicare evoluzioni nel SUT. Per esempio, se la copertura del codice diminuisce, significa molto probabilmente che sono state aggiunte funzionalità al SUT, ma non sono stati aggiunti i corrispondenti casi di test alla suite di test automatizzati.

Metriche di scripting dello strumento
Ci sono molte metriche che possono essere usate per monitorare lo sviluppo di script di automazione. La maggior parte di queste sono simili alle metriche per il codice sorgente del SUT. Le linee di codice (LOC) e la complessità ciclomatica possono essere usate per evidenziare script troppo grandi o complessi (in tal caso è necessario suggerire la possibile riprogettazione).

Il rapporto tra i commenti e le istruzioni eseguibili può essere usato per dare una possibile indicazione dell'entità della documentazione e dei commenti degli script. Il numero di non conformità agli standard di scripting può dare un'indicazione di quanto tali standard siano seguiti.

Densità di difetti nel codice dell'automazione
Il codice dell'automazione non è differente dal codice del SUT, per il fatto che è software e conterrà difetti. Il codice dell'automazione non deve essere considerato meno importante del codice del SUT. Devono essere applicati buone pratiche e standard di codifica e i risultati devono essere monitorati da metriche come la densità dei difetti del codice. Queste metriche saranno più facili da raccogliere con il supporto di un sistema di gestione della configurazione.

Velocità ed efficienza dei componenti della TAS
Differenze nel tempo impiegato per eseguire gli stessi passi dei test nello stesso ambiente possono indicare un problema nel SUT. Se il SUT non esegue la stessa funzionalità nello stesso tempo, è necessario investigare. Ciò può indicare una variabilità del sistema che non è accettabile e che potrebbe peggiorare con un carico maggiore. La TAS deve avere prestazioni tali da non mascherare le prestazioni

© International Software Testing Qualifications Board

del SUT. Se le prestazioni sono un requisito critico per il SUT, allora la TAS deve essere progettata in modo da tenerne conto.

Metriche di tendenza
Per molte di queste metriche sono le tendenze (cioè il modo in cui le misure cambiano nel tempo) ad essere di maggior valore aggiunto rispetto al singolo valore di una misura ad un istante specifico. Per esempio, sapere che il costo medio di manutenzione per i test automatizzati che richiedono di essere aggiornati è maggiore di quello che è stato necessario nelle precedenti due release del SUT, può innescare un'azione per determinare la causa dell'incremento ed intraprendere passi per invertire la tendenza.

Il costo delle misurazioni dovrebbe essere il più basso possibile e ciò può essere spesso ottenuto automatizzando la raccolta e il reporting.

5.2 Implementazione delle Misure

Poiché una strategia di automazione del testing ha al suo interno testware automatizzato, il testware automatizzato può essere arricchito per registrare informazioni sul suo uso. Quando l'astrazione è combinata con un testware strutturato, ogni miglioramento fatto al testware sottostante può essere utilizzato da tutti gli script di test automatizzati di più alto livello. Per esempio, la modifica del testware sottostante per permettere di registrare l'orario di inizio e fine dell'esecuzione di un test può essere sfruttata da tutti i test.

Caratteristiche dell'automazione che supportano la misura e la generazione di report
I linguaggi di scripting di molti strumenti di test supportano la misura e il reporting tramite funzionalità che possono essere usate per registrare e tracciare informazioni prima, durante e dopo l'esecuzione di singoli test, di insiemi di test e di un'intera suite di test.

Per generare report di una serie di esecuzioni di test c'è bisogno di avere a disposizione una funzionalità di analisi che tenga conto dei risultati delle precedenti esecuzioni in modo da poter evidenziare andamenti interessanti (come i cambiamenti nel tasso di successo dei test).

L'automazione del testing tipicamente richiede l'automazione sia dell'esecuzione che della verifica dei test, quest'ultima ottenuta tramite confronto di specifici elementi del risultato del test con un esito atteso predefinito. Questo confronto è in genere svolto più efficacemente tramite uno strumento di test. Deve essere valutato il livello di informazione che viene riportato come risultato di tale confronto. È importante che lo stato del test sia determinato correttamente (per es. passato, fallito). Nel caso di stato fallito, saranno richieste maggiori informazioni sulla causa del fallimento (per es. screen shot).

Distinguere fra differenze attese nell'esito effettivo e atteso di un test non è sempre banale sebbene il supporto dello strumento possa aiutare molto nel definire confronti che ignorino le differenze attese (come date e orari) ed evidenzino ogni differenza inattesa.

© International Software Testing Qualifications Board

Integrazione con strumenti di terze parti (fogli di calcolo, XML, documenti, database, strumenti di reportistica, ecc.)
Quando le informazioni dell'esecuzione di casi di test automatizzati sono usate in altri strumenti (per tracciare e generare report, per es. aggiornare la matrice di tracciabilità), è possibile fornire tali informazioni in un formato che sia adatto a questi strumenti di terze parti. Questo è spesso ottenuto attraverso funzionalità esistenti dello strumento di test (formati di export per i report) o creando report personalizzati che abbiano un output in un formato compatibile con altri programmi (".xls" per Excel, ".doc" per Word, ".html" per Web, ecc.).

Visualizzazione dei risultati (dashboard, diagrammi, grafici, ecc.)
I risultati dei test devono essere resi visibili in diagrammi. Si consideri l'uso di colori per indicare problemi nell'esecuzione dei test come i semafori per indicare l'avanzamento dell'esecuzione/automazione dei test e che consentano di prendere delle decisioni informate. Il management è particolarmente interessato a riassunti visuali per vedere il risultato del test a colpo d'occhio; nel caso siano necessarie maggiori informazioni di maggior dettaglio, si deve poter accedere ad esse.

5.3 Logging della TAS e del SUT

Il logging è molto importante nella TAS, incluso il logging sia della automazione del testing stessa che del SUT. I log di test sono una sorgente che frequentemente è usata per analizzare potenziali problemi. Nel seguente paragrafo sono indicati esempi di logging del testing, suddivisi per TAS e SUT.

Il logging della TAS (se siano la TAF o i casi di test stessi a registrare le informazioni non è così importante e dipende dal contesto) deve includere quanto segue:

- Quale caso di test è attualmente in esecuzione, inclusi gli orari di inizio e fine.
- Lo stato dell'esecuzione del caso di test perché, mentre gli esiti negativi possono facilmente essere identificati nei file di log, il framework stesso deve avere tale informazione e deve visualizzarla su una dashboard. Lo stato di esecuzione di un caso di test può essere passato, fallito o errore della TAS. Il risultato di errore della TAS è usato per situazioni in cui il problema non è nel SUT.
- Dettagli del log di test ad alto livello (logging di passi significativi) inclusa l'informazione dei tempi.
- Informazioni dinamiche sul SUT (per es. memory leak) che il caso di test è stato in grado di identificare con l'ausilio di strumenti di terze parti. Risultati effettivi ed esiti negativi di tali misurazioni dinamiche devono essere registrati con il caso di test che era in esecuzione quando l'incidente è stato rilevato.
- Nel caso di test di affidabilità/stress (dove sono eseguiti numerosi cicli) deve essere registrato un contatore in modo che possa essere facilmente determinato quante volte i casi di test sono stati eseguiti.
- Quando i casi di test hanno parti casuali (per es. parametri casuali o passi casuali nel testing di macchine a stati), devono essere registrati i numeri casuali e le scelte casuali.
- Tutte le azioni che un caso di test esegue devono essere registrate in modo tale che il file di log (o parti di esso) possa essere riletto per rieseguire il test con esattamente gli stessi passi e la stessa tempistica. Questo è utile per controllare la riproducibilità di un esito negativo e per catturare informazioni aggiuntive. Possono anche essere registrate le informazioni sulle azioni del caso di test sul SUT stesso da usare quando si riproducono problemi identificati dagli utenti

© International Software Testing Qualifications Board

(l'utente esegue lo scenario, le informazioni di log sono catturate e possono successivamente essere rieseguite dal team di sviluppo quando effettua l'analisi del problema).
- Schermate e altre immagini catturate possono essere salvate durante l'esecuzione dei test per un successivo utilizzo durante l'analisi degli esiti negativi.
- Ogni volta che un caso di test va incontro ad un esito negativo, la TAS deve far sì che tutte le informazioni necessarie per analizzare il problema siano disponibili/salvate, come anche ogni informazione, se possibile, circa la continuazione del testing. Tutti i crash dump e stack trace associati devono essere salvati dalla TAS in un posto sicuro. Anche ogni file di log che potrebbe essere sovrascritto (sono spesso usati buffer ciclici per i file di log del SUT) deve essere copiato in un luogo dove sarà disponibile per analisi successive.
- L'uso dei colori può aiutare a distinguere diversi tipi di informazioni registrate (per es. errori in rosso, informazioni di avanzamento in verde).

Logging del SUT:

- Quando il SUT identifica un problema, tutte le informazioni necessarie per analizzarlo devono essere registrate, inclusi date e orari, sorgente origine del problema, messaggi di errore, ecc.
- Il SUT può registrare tutte le interazioni utente (direttamente tramite l'interfaccia utente disponibile, ma anche tramite le interfacce di rete, ecc.). In questo modo i problemi identificati dagli utenti possono essere analizzati appropriatamente e lo sviluppo può cercare di riprodurli.
- All'avvio del sistema, devono essere registrate su un file le informazioni di configurazione, che comprendono le differenti versioni di software/firmware, la configurazione del SUT, la configurazione del sistema operativo, ecc.

Tutte le differenti informazioni di logging devono essere facilmente ricercabili. Un problema identificato nel file di log dalla TAS deve essere facilmente identificato nel file di log del SUT e viceversa (con o senza l'uso di strumenti aggiuntivi). Sincronizzare i vari log con un timestamp facilita la correlazione di cosa accade quando viene scoperto un errore.

5.4 Reportistica dell'Automazione del Testing

I log del test danno informazioni dettagliate su passi di esecuzione, azioni e risposte di un caso di test e/o di una suite di test. Comunque, i log da soli non possono fornire una buona panoramica del risultato complessivo dell'esecuzione. Per questo, è necessario avere a disposizione funzionalità di reportistica. Dopo ciascuna esecuzione della suite di test, deve essere creato e pubblicato un report conciso. Un componente generatore di report riusabile deve poter essere utilizzato a questo scopo.

Contenuto dei report
I report di esecuzione dei test devono contenere un sommario che dia una panoramica dei risultati dell'esecuzione, del sistema che si sta testando e dell'ambiente in cui i test sono stati eseguiti, che sia appropriata per ciascuno degli attori a cui il report è rivolto.

È necessario sapere quali test sono falliti e le ragioni degli esiti negativi. Per rendere più facile l'analisi dei problemi, è importante conoscere la storia delle esecuzioni del test e chi è il suo responsabile (in genere la persona che lo ha creato o modificato per ultima). La persona responsabile deve investigare la causa dell'esito negativo, produrre report dei problemi ad esso collegati, seguire la correzione dei problemi e controllare che tale correzione sia stata correttamente implementata.

© International Software Testing Qualifications Board

La reportistica è anche usata per diagnosticare ogni esito negativo dei componenti della TAF (si veda il Capitolo 7).

Pubblicare i report
I report devono essere pubblicati verso ogni persona interessata ai risultati dell'esecuzione. Possono essere caricati su un sito web, inviati ad una mailing list o caricati su un altro strumento come uno strumento di gestione del testing. Dal lato pratico, è più probabile che gli interessati al risultato dell'esecuzione guarderanno il report e lo analizzeranno se viene data loro una possibilità di sottoscrizione e possano ricevere il report per e-mail.

Una scelta per identificare le parti problematiche del SUT, è mantenere una storia dei report, così che possano essere raccolte statistiche su casi di test e suite di test con frequenti regressioni.

© International Software Testing Qualifications Board

6. Transizione dal Testing Manuale ad un Ambiente Automatizzato - 120 min.

Termini
testing confermativo, testing di regressione

Obiettivi di apprendimento su Transizione dal Testing Manuale ad un Ambiente Automatizzato

6.1 Criteri per l'Automazione

ALTA-E-6.1.1 (K3) Applicare criteri per determinare l'idoneità dei test all'automazione
ALTA-E-6.1.2 (K2) Comprendere i fattori in gioco nella transizione dal testing manuale a quello automatizzato

6.2 Identificare i passi necessari per implementare l'Automazione per il Testing di Regressione

ALTA-E-6.2.1 (K2) Spiegare i fattori da considerare nell'implementazione del testing di regressione automatizzato

6.3 Fattori da considerare quando si implementa l'Automazione per il Testing di Nuove Funzionalità

ALTA-E-6.3.1 (K2) Spiegare i fattori da considerare nell'implementazione dell'automazione per il testing di nuove funzionalità

6.4 Fattori da considerare quando si implementa l'Automazione del Testing Confermativo

ALTA-E-6.4.1 (K2) Spiegare i fattori da considerare nell'implementazione del testing confermativo automatizzato

© International Software Testing Qualifications Board

6.1 Criteri per l'Automazione

Tradizionalmente, le varie organizzazioni hanno sviluppato casi di test manuali. Quando si decide di migrare verso un ambiente di test automatizzato, si deve valutare lo stato corrente dei test manuali e determinare l'approccio più efficace per automatizzare tali asset di testing. La struttura esistente di un test manuale può o non può essere adatta all'automazione, e può quindi essere necessaria una completa riscrittura del test per supportare l'automazione. In alternativa, le componenti più importanti dei test manuali esistenti (per es. dati di input, risultati attesi, cammini di navigazione) possono essere estratte dai test manuali esistenti e riusate per l'automazione, Una strategia di test manuale che tenga conto dell'automazione permetterà di avere test la cui struttura faciliti la migrazione verso l'automazione.

Non tutti i test possono o devono essere automatizzati e talvolta la prima iterazione di un test può essere manuale. Perciò ci sono due aspetti della transizione da considerare: la conversione iniziale all'automazione dei test manuali esistenti e la successiva transizione verso l'automazione dei nuovi test manuali.

È da notare anche che certi tipi di test possono solo essere eseguiti (efficacemente) in modo automatizzato, per es. test di affidabilità, di stress o prestazionali.

Con l'automazione del testing è possibile testare applicazioni e sistemi senza un'interfaccia utente. In questo caso il testing può essere effettuato a livello di test di integrazione per mezzo delle interfacce del software. Questi tipi di casi di test possono anche essere eseguiti manualmente (usando comandi inviati manualmente per sollecitare le interfacce), ma ciò può non essere pratico. Per esempio, con l'automazione è possibile inserire messaggi in un sistema di code di messaggi. In questo modo il testing può iniziare prima (e può identificare i difetti prima), quando il testing manuale non è ancora possibile.

Prima di iniziare uno sforzo di testing automatizzato, è necessario considerare l'applicabilità e la fattibilità di creare test automatizzati rispetto a quelli manuali. I criteri di idoneità possono includere:

- Frequenza d'uso
- Complessità dell'automazione
- Compatibilità dello strumento di supporto
- Maturità del processo di test
- Idoneità dell'automazione per la fase del ciclo di vita del prodotto software
- Sostenibilità dell'ambiente automatizzato
- Controllabilità del SUT

Ciascuno di questi criteri è spiegato in maggiore dettaglio di seguito.

Frequenza d'uso
Quanto spesso un test deve essere eseguito è un criterio che incide sulla decisione o meno di automatizzare. I test che sono eseguiti più regolarmente, come parte di un ciclo di test di una release principale o minore, sono i migliori candidati per l'automazione in quanto saranno usati frequentemente. Come regola generale, più grande è il numero di release di un'applicazione – e quindi dei corrispondenti cicli di test – maggiori sono i benefici di automatizzare i test. Quando i test funzionali sono automatizzati, possono essere usati nelle release successive come parte del testing di regressione. I test automatizzati usati nel testing di regressione forniranno un più alto ritorno di investimento (ROI) e di mitigazione del rischio per il codice esistente.

© International Software Testing Qualifications Board

Se uno script di test è eseguito una volta all'anno e il SUT cambia all'interno dell'anno, può non essere praticabile o efficiente creare un test automatizzato. Il tempo che potrebbe essere necessario per adattare i test su base annua per allinearli al SUT, potrebbe essere speso meglio eseguendo i test manualmente.

Complessità dell'automazione
Nei casi in cui sia necessario testare un sistema complesso, si può trarre enorme beneficio dall'automazione risparmiando al tester manuale il difficile compito di ripetere passi complessi che sono tediosi, richiedono molto tempo e sono soggetti ad errore nell'essere eseguiti.

Comunque, certi script di test possono essere difficili e costosi da automatizzare. Esistono vari fattori che possono influenzare ciò, tra cui: un SUT che non è compatibile con le soluzioni di test automatizzato disponibili; la necessità di produrre una considerevole quantità di codice e sviluppare chiamate ad API per automatizzare i test; la molteplicità di sistemi che devono essere coperti dall'esecuzione del test; l'interazione con interfacce esterne e/o con sistemi proprietari; alcuni aspetti del testing di usabilità; la quantità di tempo necessaria per testare gli script automatici, ecc.

Compatibilità dello strumento di supporto
C'è un ampio ventaglio di piattaforme di sviluppo utilizzate per creare applicazioni. La sfida per il tester è sapere quali strumenti di test sono disponibili (se ci sono) per supportare una data piattaforma e fino a che punto la piattaforma è supportata. Le organizzazioni usano una varietà di strumenti di test, inclusi quelli commerciali, open source e sviluppati in casa. Ogni organizzazione avrà necessità e risorse diverse per supportare gli strumenti di test. I venditori commerciali tipicamente forniscono un supporto a pagamento e, nel caso di leader di mercato, di solito hanno un ecosistema di esperti che può fornire assistenza per l'implementazione degli strumenti di test. Per gli strumenti open source si può avere supporto tramite i forum online con cui gli utenti possono ottenere informazioni e porre domande. Per gli strumenti di test sviluppati internamente ci si rivolge allo staff presente per ricevere supporto.

I problemi di compatibilità degli strumenti di test non devono essere sottostimati. Imbarcarsi in un progetto di automazione del testing senza comprendere pienamente il livello di compatibilità fra gli strumenti di test e il SUT può portare a risultati disastrosi. Anche se la maggior parte dei test del SUT possono essere automatizzati, ci potrebbe essere la situazione in cui i più critici possono non esserlo.

Maturità del processo di test
Per implementare efficacemente l'automazione all'interno di un processo di test, tale processo deve essere strutturato, disciplinato e ripetibile. L'automazione porta un intero processo di sviluppo dentro il processo di testing esistente e ciò richiede di gestire il codice dell'automazione e i componenti collegati.

Idoneità dell'automazione per la fase del ciclo di vita del prodotto software
Un SUT ha un ciclo di vita che può variare da anni a decenni. Quando lo sviluppo di un sistema inizia, il sistema cambia e si espande per risolvere difetti e aggiungere raffinamenti per venire incontro ai bisogni degli utenti finali. Nelle prime fasi di sviluppo di un sistema, i cambiamenti possono essere troppo rapidi per implementare una soluzione di testing automatizzato. Mentre le schermate e i controlli sono ottimizzati e aggiornati, creare l'automazione in un ambiente che cambia dinamicamente può richiedere continue rilavorazioni, il che non né efficiente né efficace. Sarebbe come cercare di cambiare una gomma ad una macchina in movimento; è meglio aspettare che la macchina si fermi. Per grandi sistemi in ambienti di sviluppo sequenziale, il momento migliore per iniziare l'implementazione dei test automatizzati è quando un sistema è stabilizzato e include le funzionalità più importanti.

© International Software Testing Qualifications Board

Nel tempo, i sistemi raggiungono la fine del loro ciclo di vita e sono o eliminati o riprogettati per usare tecnologie più nuove e più efficienti. L'automazione non è raccomandata per un sistema vicino alla fine del suo ciclo di vita poiché sarà di poco valore intraprendere un'iniziativa con una vita così breve. Comunque, per sistemi che stanno per essere riprogettati usando un'architettura differente, ma preservando le funzionalità esistenti, un ambiente di test automatizzato che definisca i dati sarà ugualmente utile nei vecchi e nei nuovi sistemi. In questo caso, il riuso dei dati di test sarebbe possibile e si renderebbe necessario ricodificare l'ambiente automatizzato per ottenere la compatibilità con la nuova architettura.

Sostenibilità dell'ambiente automatizzato
Un ambiente di test per l'automazione deve essere flessibile e adattabile ai cambiamenti che avverranno al SUT nel tempo. Ciò include l'abilità di diagnosticare rapidamente e correggere i problemi dell'automazione, la facilità con cui i componenti dell'automazione possono essere manutenuti e la facilità con cui nuove funzionalità possono essere aggiunte all'ambiente automatizzato. Questi attributi sono una parte integrale del progetto complessivo e dell'implementazione della gTAA.

Controllabilità del SUT (precondizioni, setup e stabilità)
Il TAE deve identificare le caratteristiche di controllo e visibilità nel SUT che aiuteranno nella creazione di efficaci test automatizzati. Altrimenti l'automazione del testing si baserà solo su interazioni con la UI, con la conseguenza di avere una soluzione di automazione del testing meno mantenibile. Si veda il paragrafo 2.3 su Progettare per la Testabilità e l'Automazione per maggiori informazioni.

Pianificazione tecnica a supporto dell'analisi del ROI
L'automazione del testing può fornire benefici a vari livelli a un team di test. Tuttavia, all'implementazione di un'efficace soluzione di testing automatizzato è associato un significativo livello di sforzo e di costo. Prima di sostenere lo sforzo e il tempo per sviluppare test automatizzati, deve essere condotta una stima per valutare quali potrebbero essere i benefici complessivi attesi e potenziali e i risultati di implementare l'automazione del testing. Una volta che ciò è stato determinato, devono essere definite le attività necessarie per realizzare tale piano e devono essere determinati i costi associati per calcolare il ROI.

Per preparare adeguatamente la transizione ad un ambiente automatizzato, devono essere considerate le seguenti aree:

- Disponibilità di strumenti nell'ambiente di test per l'automazione del testing
- Correttezza di dati e casi di test
- Ambito dello sforzo di automazione del testing
- Educazione del team di test al cambiamento di paradigma
- Ruoli e responsabilità
- Cooperazione fra sviluppatori e test automation engineer
- Sforzo parallelo
- Reporting dell'automazione del testing

Disponibilità di strumenti nell'ambiente di test per l'automazione del testing
Gli strumenti di test selezionati devono essere installati e deve esserne confermato il funzionamento nell'ambiente del laboratorio di test. Ciò può richiedere di installare service pack e aggiornamenti di release, di scegliere la configurazione di installazione appropriata – inclusi i componenti aggiuntivi – necessaria per supportare il SUT e di assicurare che la TAS funzioni correttamente nell'ambiente del laboratorio di test, oltre che nell'ambiente di sviluppo dell'automazione

© International Software Testing Qualifications Board

Correttezza di dati e casi di test
La correttezza e la completezza dei dati di test e dei casi di test manuali sono necessarie per assicurare che l'uso dell'automazione fornirà risultati prevedibili. Per i test che sono eseguiti con l'automazione è necessario che siano indicati in forma esplicita i dati per l'input, la navigazione, la sincronizzazione e la validazione.

Ambito dello sforzo di automazione del testing
Per mostrare i primi successi nell'automazione e ottenere riscontri su problemi tecnici che possano impattarne l'avanzamento, si raccomanda di iniziare con un ambito limitato che faciliterà anche le future attività di automazione. Un progetto pilota può mirare ad un'area di una funzionalità del sistema che sia rappresentativa dell'interoperabilità dell'intero sistema. Le "lesson learned" dal pilota aiuteranno a tarare le future stime e pianificazioni e ad identificare le aree che richiedono risorse tecniche specializzate. Un progetto pilota fornisce un modo veloce di mostrare i primi successi dell'automazione, ottenendo un ulteriore supporto da parte del management.

Per aiutare in questo, i casi di test da automatizzare devono essere selezionati in modo saggio. Vanno scelti i casi che richiedono un piccolo sforzo per essere automatizzati, ma forniscono un alto valore aggiunto. Test di regressione automatici o smoke test possono essere implementati e aggiungere un valore considerevole, poiché questi test sono normalmente eseguiti molto spesso, anche giornalmente. Un altro buon candidato con cui iniziare è il testing di affidabilità. Questi test sono spesso composti di passi che sono eseguiti ripetutamente, rivelando problemi che sono difficili da scoprire manualmente. Questi test di affidabilità necessitano di un piccolo sforzo per essere implementati e possono fornire un tempestivo valore aggiunto.

Questi progetti pilota mettono l'automazione sotto i riflettori (sforzo risparmiato per il test manuale o seri problemi identificati) ed aprono la strada per ulteriori estensioni (sforzo e denaro).

In aggiunta, deve essere data priorità ai test che sono critici per l'organizzazione, poiché questi mostreranno all'inizio il valore più grande. Comunque, all'interno di questo contesto, è importante che come parte dello sforzo per il pilota, si eviti di automatizzare i test tecnicamente più sfidanti. Altrimenti, sarà impiegato troppo sforzo per cercare di sviluppare l'automazione e ci saranno troppi pochi risultati da mostrare. Come regola generale, identificare i test che condividono caratteristiche/funzionalità con gran parte dell'applicazione fornirà lo slancio per mantenere vivo lo sforzo di automazione.

Educazione del team di test al cambiamento di paradigma
I tester possono avere diverse competenze: alcuni sono esperti del dominio provenendo dalla comunità degli utenti finali o essendo coinvolti come analisti del business, mentre altri hanno forti competenze tecniche che li rendono capaci di comprendere la sottostante architettura del sistema. Affinché il testing sia efficace è richiesto un ampio insieme di competenze. Quando il team di test passa all'automazione, i ruoli diventano più specializzati. Affinché l'automazione abbia successo diventa essenziale cambiare la composizione del team di test ed educare il team in anticipo sul cambiamento atteso aiuterà a ridurre l'ansia sui ruoli o sull'eventualità di essere considerati ridondanti. Quando indirizzata correttamente, la transizione all'automazione dovrebbe rendere ogni persona del team di test entusiasta e pronta a partecipare al cambiamento organizzativo e tecnico.

Ruoli e responsabilità
L'automazione del testing è un'attività a cui tutti possono partecipare. Tuttavia, ciò non significa che tutti debbano avere lo stesso ruolo. Progettare, implementare e manutenere un ambiente di test automatizzato sono attività di natura tecnica e come tali devono essere riservate a persone con forti competenze di programmazione e background tecnico. Il risultato di uno sforzo di sviluppo di testing

© International Software Testing Qualifications Board

automatizzato deve essere un ambiente che sia utilizzabile sia da persone tecniche che non tecniche. Per massimizzare il valore di un ambiente di testing automatizzato c'è bisogno di persone con esperienza sul dominio e con competenze sul testing, poiché sarà necessario sviluppare script di test appropriati (inclusi i corrispondenti dati di test). Questi saranno usati per pilotare l'ambiente di test automatizzato e fornire la copertura dei test desiderata. Gli esperti del dominio hanno il compito di esaminare i report e confermare la funzionalità dell'applicazione, mentre gli esperti tecnici quello di assicurare che l'ambiente automatizzato stia operando correttamente ed efficientemente. Questi esperti tecnici possono anche essere sviluppatori con un interesse nel testing. L'esperienza nello sviluppo del software è essenziale per progettare un software che sia mantenibile e questo è di estrema importanza nell'automazione del testing. Gli sviluppatori si possono concentrare sul framework di automazione dei testing o sulle librerie di test. L'implementazione dei casi di test deve rimanere ai tester.

Cooperazione fra sviluppatori e TAE (test automation engineer)
Il successo dell'automazione del testing richiede il coinvolgimento del team di sviluppo del software e dei tester. Sviluppatori e tester dovranno lavorare molto più a stretto contatto per l'automazione del testing in modo tale che gli sviluppatori possano fornire supporto personale e informazioni tecniche sui loro metodi e strumenti di sviluppo. I TAE possono sollevare preoccupazioni sulla testabilità della progettazione del sistema e del codice sviluppato. Questo accadrà specialmente nel caso in cui gli standard non sono seguiti o se gli sviluppatori usano librerie/oggetti strani, fatti in casa o anche nuovissimi. Per esempio, gli sviluppatori potrebbero scegliere un controllo GUI di una terza parte che potrebbe non essere compatibile con lo strumento di test selezionato. In fine, un team di gestione progetti dell'organizzazione deve avere chiara comprensione dei tipi di ruoli e responsabilità richiesti per uno sforzo di automazione di successo.

Sforzo parallelo
Come parte delle attività di transizione, molte organizzazioni creano un team parallelo per iniziare il processo di automazione degli script manuali di test esistenti. I nuovi script automatizzati sono poi incorporati nel processo di testing, rimpiazzando gli script manuali. Tuttavia, prima di ciò, è spesso raccomandato fare un controllo e validare che gli script automatizzati eseguano gli stessi test e validazioni degli script manuali che stanno sostituendo.

In molte occasioni, sarà fatta una valutazione degli script di test prima della conversione all'automazione. Come risultato di tale valutazione, potrà essere identificata la necessità di ristrutturare gli script di test manuali esistenti per un approccio più efficiente ed efficace all'automazione.

Reporting dell'automazione del testing
Ci sono vari report che possono essere generati automaticamente da una TAS. Questi includono lo stato pass/fail dei singoli script o dei passi all'interno degli script, le statiche generali sull'esecuzione dei test e le prestazioni complessive della TAS. È ugualmente importante avere visibilità della corretta operatività della TAS così che ogni risultato specifico dell'applicazione che sia evidenziato possa essere veramente accurato e completo (Si veda il capitolo 7: Verificare la TAS).

© International Software Testing Qualifications Board

6.2 Identificare i passi necessari ad implementare l'Automazione per il Testing di Regressione

Il testing di regressione fornisce una grande opportunità per usare l'automazione. Un test di regressione cresce quando i casi di test funzionali di oggi diventano i test di regressione di domani. È solo una questione di tempo prima che il numero di test di regressione diventi più grande del tempo e delle risorse disponibili per un tradizionale team di test manuale.

Nello sviluppare i passi per prepararsi ad automatizzare i test di regressione, ci si deve porre una serie di domande:

- Quanto frequentemente i test devono essere eseguiti?
- Quale è il tempo di esecuzione per ciascun test e per la suite di regressione?
- Esiste sovrapposizione funzionale tra i test?
- I test condividono dei dati?
- Esistono dipendenze tra i test?
- Quali precondizioni sono richieste prima dell'esecuzione dei test?
- Quale è la percentuale di copertura del SUT da parte dei test?
- Attualmente l'esecuzione dei test avviene senza esiti negativi?
- Cosa fare quando i test di regressione richiedono troppo tempo di esecuzione?

Ognuna di queste domande è spiegata in maggiore dettaglio di seguito.

Frequenza di esecuzione dei test
I test che sono eseguiti spesso come parte del testing di regressione sono i migliori candidati per l'automazione. Questi test sono già stati sviluppati, sollecitano funzionalità note del SUT e il loro tempo di esecuzione sarà ridotto estremamente tramite l'uso dell'automazione.

Tempo di esecuzione dei test
Il tempo necessario per eseguire ciascun test o un'intera suite di test è un parametro importante nel valutare il valore di implementare il testing automatizzato nell'ambito del testing di regressione. Un'opzione è iniziare ad implementare l'automazione per i test che necessitano di più tempo per l'esecuzione. Ciò consentirà a ciascun test di essere eseguito più velocemente ed efficientemente, permettendo anche di aggiungere cicli aggiuntivi di esecuzione automatizzata dei test di regressione. Il beneficio consisterà in feedback aggiuntivi e più frequenti sulla qualità del SUT e in ridotti rischi di rilascio.

Sovrapposizione funzionale
Quando si automatizzano test di regressione esistenti, è una buona pratica identificare ogni sovrapposizione funzionale che esiste tra i casi di test e, dove possibile, ridurre tali sovrapposizioni nei test automatizzati corrispondenti. Ciò porterà ulteriore efficienza nel tempo di esecuzione dei test automatizzati, che sarà significativa quanti più casi di casi di test automatizzati saranno eseguiti. Spesso, i test sviluppati usando l'automazione avranno una nuova struttura poiché dipendono da componenti riutilizzabili e repository di dati condivisi. Non è insolito decomporre i test manuali esistenti in diversi test automatizzati più piccoli. Ugualmente, il consolidamento di diversi test manuali in un test automatizzato

© International Software Testing Qualifications Board

più grande può essere una soluzione appropriata. I test manuali devono essere valutati individualmente, e come gruppo, in modo che possa essere sviluppata una strategia di conversione efficace.

Condivisione dei dati
I test spesso condividono i dati. Questo può avvenire quanto i test usano lo stesso insieme di dati per eseguire diverse funzionalità del SUT. Un esempio di ciò potrebbe essere il caso di test "A" che verifica il tempo di ferie disponibile per un impiegato, mentre il caso di test "B" potrebbe verificare i corsi che l'impiegato segue come parte degli obiettivi di sviluppo della carriera. Ciascun caso di test usa lo stesso impiegato, ma verifica parametri differenti. In un ambiente di test manuale, i dati dell'impiegato sarebbero tipicamente duplicati molte volte per ciascun caso di test manuale che controlla i dati dell'impiegato. Invece, in un test automatizzato, i dati che sono condivisi devono – dove possibile e realizzabile – essere memorizzati e acceduti da una singola sorgente per evitare duplicazioni o introdurre errori.

Interdipendenza dei test
Quando si eseguono scenari di test di regressione complessi, un test può aver una dipendenza da uno o più test. Questa evenienza può essere abbastanza comune e può accadere, per esempio, per un nuovo "ID Ordine" che viene creato come risultato di un passo di un test. Test successivi possono dover verificare che: a) il nuovo ordine è correttamente presente a sistema, b) le modifiche all'ordine siano possibili, c) la cancellazione dell'ordine ha esito positivo. In ciascun caso, il valore del "ID ordine" che è creato dinamicamente nel primo test deve essere catturato per essere riusato nei test successivi. Questo requisito può essere soddisfatto tramite un'adeguata progettazione della TAS.

Precondizioni dei test
Spesso un test non può essere eseguito prima di impostare delle condizioni iniziali. Queste condizioni possono includere la scelta della base dati corretta o dell'insieme dati di test con cui testare, o l'impostazione di valori iniziali o parametri. Molti di questi passi di inizializzazione che sono necessari per impostare le precondizioni di un test possono essere automatizzati. Questo consente una soluzione più affidabile e sicura quando tali passi non possono essere omessi prima di eseguire i test. Quando i test di regressione sono convertiti all'automazione, queste precondizioni devono essere parte del processo di automazione.

Copertura del SUT
Ogni volta che i test sono eseguiti, sono sollecitate parti delle funzionalità di un SUT. Per verificare la qualità dell'intero SUT, i test devono essere progettati per ottenere la copertura più estesa e più profonda possibile. In aggiunta, possono essere usati degli strumenti di copertura del codice per monitorare l'esecuzione dei test automatizzati con l'obiettivo di aiutare a quantificare l'efficacia dei test. Attraverso il testing di regressione automatizzato, nel tempo ci si attende che i test aggiunti porteranno ad un aumento della copertura. Misurare questo aumento fornisce un mezzo efficace per quantificare il valore dei test stessi.

Test eseguibili
Prima di convertire un test di regressione manuale in un test automatizzato, è importante verificare che il test manuale funzioni correttamente. Ciò fornisce un punto di partenza corretto per assicurare che la sua conversione in un test di regressione automatizzato abbia successo. Se il test manuale non si esegue correttamente - o perché è scritto male, usa dati non validi, non è aggiornato o non sincronizzato con la versione corrente del SUT o a causa di un difetto del SUT – convertirlo in un test automatizzato prima di

© International Software Testing Qualifications Board

comprendere e/o risolvere la causa del fallimento creerà un test automatico non funzionante, che sarà inutile e non produttivo.

Grandi insiemi di test di regressione
L'insieme dei test di regressione per un SUT può diventare talmente grande da non poter esser eseguito completamente in una notte o in un weekend. In questo caso, l'esecuzione concorrente dei casi di test è una possibilità se sono disponibili diversi SUT (per applicazioni PC ciò probabilmente non pone alcun problema, ma quando il SUT consiste in un aeroplano o un razzo spaziale è una storia differente). I SUT possono essere scarsi e/o costosi tanto da rendere la concorrenza un'opzione non realistica. In questo caso, una possibilità può essere quella di eseguire solo parte dei test di regressione. Nel tempo (settimane) può essere eseguito realmente l'intero insieme dei test. La scelta di quale parte della suite di test di regressione eseguire può anche essere basata sull'analisi del rischio (quale parte del SUT è stata cambiata di recente?).

6.3 Fattori da considerare quando si implementa l'Automazione per il Testing di Nuove Funzionalità

In generale è più facile automatizzare casi di test per una nuova funzionalità quando l'implementazione non è ancora finita (o meglio: non ancora iniziata). Il test engineer può usare le sue conoscenze per spiegare agli sviluppatori e agli architetti esattamente cosa sia necessario nella nuova funzionalità, affinché possa essere testata in modo efficace ed efficiente dalla soluzione di automazione del testing.

Quando nuove funzionalità sono introdotte in un SUT, i tester devono sviluppare nuovi test per queste nuove funzionalità e i relativi requisiti. Il TAE deve sollecitare dei riscontri da parte dei progettisti dei test con gli esperti di dominio per determinare se la TAS corrente soddisferà quanto richiesto per le nuove funzionalità. In questa analisi sono inclusi, senza escludere altri, l'approccio che attualmente viene usato, gli strumenti di sviluppo di terze parti, gli strumenti di test usati, ecc.

I cambiamenti alla TAS devono essere valutati rispetto ai componenti esistenti del testware in modo tale che i cambiamenti o le aggiunte siano pienamente documentati e non influenzino il comportamento (o le prestazioni) delle funzionalità esistenti della TAS.

Se una nuova funzionalità è implementata con, per esempio, una diversa classe di un oggetto, può essere necessario fare aggiornamenti o aggiunte ai componenti del testware. Inoltre, deve essere valutata la compatibilità con gli strumenti di test esistenti e, dove necessario, identificate nuove soluzioni. Per esempio, se si usa un approccio al testing keyword-driven, può essere necessario sviluppare keyword aggiuntive o modificare/espandere le keyword esistenti per adattarle alle nuove funzionalità.

Ci può essere la necessità di valutare strumenti di test aggiuntivi per supportare il nuovo ambiente nel quale la nuova funzionalità è realizzata. Per esempio, un nuovo strumento di test potrebbe essere necessario se lo strumento di test attuale supporta solo HTML.

Nuovi requisiti di test possono influenzare i test automatizzati esistenti e i componenti del testware. Quindi, prima di fare cambiamenti, i test automatizzati esistenti devono essere eseguiti sul SUT nuovo/aggiornato per verificare e registrare eventuali modifiche al corretto funzionamento dei test automatizzati esistenti. Ciò deve comprendere la mappatura delle interdipendenze con gli altri test. Ogni

© International Software Testing Qualifications Board

nuovo cambiamento nella tecnologia richiederà di valutare i componenti del testware attuale (inclusi gli strumenti di test, le librerie di funzioni, le API, ecc.) e la compatibilità con l'attuale TAS.

Quando i requisiti attuali cambiano, lo sforzo di aggiornare i casi di test che verificano tali requisiti deve essere parte della pianificazione del progetto (work breakdown structure). La tracciabilità dai requisiti ai casi di test indicherà quali casi di test devono essere aggiornati. Questi aggiornamenti devono essere parte del piano complessivo.

Infine, è necessario determinare se la TAS esistente continuerà a soddisfare le esigenze attuali del SUT. Le implementazioni tecniche sono ancora valide o è richiesta una nuova architettura e questa può essere realizzata estendendo le attuali funzionalità?

Quando si introduce una nuova funzionalità, i test engineer hanno l'opportunità di assicurare che la nuova funzionalità sia testabile. Durante la fase di progettazione, si deve tener conto del testing pianificando di fornire le interfacce di test che possano essere usate dai linguaggi di scripting o dallo strumento di automazione del testing per verificare la nuova funzionalità. Si veda il paragrafo 2.3, Progettare per la Testabilità e l'Automazione, per maggiori informazioni.

6.4 Fattori da considerare quando si implementa l'Automazione del Testing Confermativo

Il testing confermativo è eseguito a seguito di una correzione al codice che risolve un difetto. Un tester tipicamente esegue i passi necessari per replicare il difetto per verificare che il difetto non esiste più.

I difetti hanno la capacità di riprodursi nelle release successive (ciò può indicare un problema di gestione della configurazione) e quindi i test confermativi sono candidati primari per l'automazione. L'uso dell'automazione aiuterà a ridurre il tempo di esecuzione per il testing confermativo. Il testing confermativo può essere aggiunto all'attuale insieme dei test di regressione automatizzata e completarlo.

Il testing confermativo automatizzato tipicamente ha un ristretto ambito di funzionalità. L'implementazione può avvenire in ogni momento in cui un difetto viene identificato e i passi per replicarlo sono ben identificati. I test confermativi automatizzati possono essere incorporati in una suite standard di regressione automatizzata o, dove più pratico, inclusi nei test automatizzati esistenti. Con entrambi gli approcci, il valore di automatizzare il testing confermativo dei difetti rimane. Tracciare i test confermativi automatizzati permette di avere dei report aggiuntivi del tempo e del numero di cicli spesi per risolvere i difetti.

In aggiunta al testing confermativo è necessario il testing di regressione per assicurare che nuovi difetti non siano stati introdotti come effetto collaterale della correzione dei difetti. Un'analisi di impatto può essere necessaria per determinare l'ampiezza appropriata del testing di regressione.

© International Software Testing Qualifications Board

7. Verificare la TAS - 120 min.

Termini
verifica

Obiettivi di apprendimento su Verificare la TAS

7.1 Verificare i Componenti dell'Ambiente di Test Automatizzato

ALTA-E-7.1.1 (K3) Verificare la correttezza di un ambiente di test automatizzato incluso il setup dello strumento di test

7.2 Verificare la Suite di Test Automatizzati

ALTA-E-7.2.1 (K3) Verificare il corretto comportamento per un dato script di test automatizzato e/o una suite di test automatizzati

© International Software Testing Qualifications Board

7.1 Verificare i Componenti dell'Ambiente di Test Automatizzato

Il team di automazione del testing deve verificare che l'ambiente di test automatizzato operi nel modo atteso. Questi controlli sono fatti, per esempio, prima di iniziare il testing automatizzato.

Ci sono vari passi che possono essere compiuti per verificare i componenti dell'ambiente di test automatizzato. Ciascuno di questi è spiegato in maggiore dettaglio di seguito:

Installazione, setup, configurazione e personalizzazione dello strumento di test
La TAS è composta di molti componenti. Ciascuno di questi deve essere tenuto in considerazione per assicurare delle prestazioni affidabili e ripetibili. Il nucleo della TAS contiene i componenti eseguibili, le corrispondenti librerie funzionali, i dati di supporto e i file di configurazione. Il processo di configurazione di una TAS può variare dall'uso di script di installazione automatizzati fino al posizionamento manuale dei file nelle corrispondenti cartelle. Gli strumenti di test, come i sistemi operativi e altre applicazioni, hanno regolarmente dei service pack o possono avere aggiunte opzionali od obbligatorie per assicurare la compatibilità con ogni dato ambiente del SUT.

L'installazione automatizzata (o la copia) da un repository centrale ha dei vantaggi. Essa permette di garantire che i test su SUT differenti siano stati eseguiti con la stessa versione della TAS e la stessa configurazione della TAS, dove questa sia appropriata. Aggiornamenti alla TAS possono essere fatti per mezzo del repository. L'uso di repository e il processo per aggiornare la TAS ad una nuova versione devono essere gli stessi degli strumenti di sviluppo standard.

Script di test con esiti di esecuzione positivi (pass) e negativi (fail) noti
Quando casi di test per cui è noto un esito positivo di esecuzione (pass) falliscono, è immediatamente chiaro che qualche cosa di fondamentale è sbagliato e deve essere corretto quanto prima. Viceversa, quando casi di test per cui è noto un esito negativo di esecuzione (fail) passano, c'è bisogno di identificare il componente che non ha funzionato correttamente. È importante controllare la corretta generazione dei file di log e delle metriche delle prestazioni come pure setup e teardown automatizzati dei casi/script di test. È anche utile eseguire alcuni test dei differenti tipi e livelli di test (test funzionali, test prestazionali, testi di componente, ecc.). Questo deve essere eseguito anche a livello di framework.

Ripetibilità nel setup/teardown dell'ambiente di test
Una TAS sarà implementata su una varietà di sistemi e server. Per assicurare che la TAS funzioni correttamente in ciascun ambiente, è necessario avere un approccio sistematico per caricare e scaricare la TAS da ogni dato ambiente. Questo è ottenuto con successo nel caso in cui la compilazione e la ricompilazione della TAS non danno alcuna differenza distinguibile quando operano all'interno dei diversi ambienti o tra un ambiente e l'altro. La gestione della configurazione dei componenti della TAS assicura che una data configurazione possa essere creata in modo affidabile.

Configurazione dell'ambiente e dei componenti di test
Comprendere e documentare i vari componenti che compongono la TAS fornisce la conoscenza necessaria per quegli aspetti della TAS che possono essere impattati o richiedere cambiamenti quando l'ambiente del SUT cambia.

© International Software Testing Qualifications Board

Connettività verso sistemi/interfacce interni ed esterni
Una volta che una TAS è installata in un dato ambiente del SUT e prima del suo reale utilizzo sul SUT, devono essere eseguiti una serie di controlli o precondizioni per assicurare che la connettività verso sistemi interni ed esterni, interfacce, ecc., sia disponibile. Impostare le precondizioni per l'automazione è essenziale per assicurare che la TAS sia stata installata e configurata correttamente.

Intrusività degli strumenti per il testing automatizzato
La TAS spesso è fortemente accoppiata con il SUT. Questo avviene per progettazione, affinché ci sia un alto livello di compatibilità specialmente per quanto concerne le interazioni al livello di GUI. Tuttavia, questa stretta integrazione può anche avere effetti negativi. Questi possono includere: un SUT si comporta in modo differente quando la TAS è installata nell'ambiente del SUT; il SUT si comporta in modo differente rispetto a quando è usato manualmente; le prestazioni del SUT sono influenzate dalla TAS installata nel suo ambiente o quando si esegue la TAS sul SUT.

Il livello di intrusione/intrusività differisce a seconda dell'approccio all'automazione del testing scelto. Per esempio:

- Quando ci si interfaccia con il SUT da interfacce esterne, il livello di intrusione sarà molto basso. Interfacce esterne possono essere segnali elettronici (per interruttori fisici), segnali USB per dispositivi USB (come le tastiere). Con questo approccio l'utente finale è simulato nel miglior modo. In questo approccio il software del SUT non è cambiato per nulla per scopi di testing. Il comportamento e i tempi di risposta del SUT non sono influenzati dall'approccio di test. Interfacciarsi col SUT in questo modo può essere molto complesso. Potrebbero essere necessari hardware dedicati, linguaggi di descrizione dell'hardware per interfacciarsi con il SUT, ecc. Per sistemi solo software questo non è un tipico approccio, ma per prodotti con software embedded questo approccio è più comune.
- Quando ci si interfaccia con il SUT a livello di GUI, l'ambiente del SUT è adattato per iniettare comandi di UI ed estrarre informazioni necessarie ai casi di test. Il comportamento del SUT non è cambiato direttamente, ma i tempi di risposta sono influenzati e questo può avere un impatto sul comportamento. Il livello di intrusione è più alto che nel caso precedente, ma interfacciarsi con il SUT in questo modo è meno complesso. Spesso possono essere usati strumenti commerciali off-the-shelf per questo tipo di automazione.
- Ci si può interfacciare con il SUT tramite interfacce di test nel software o usando interfacce esistenti già fornite dal software stesso. La disponibilità di queste interfacce (API) è una parte importante della progettazione per la testabilità. Il livello di intrusione può essere piuttosto alto in questo caso. I test automatizzati usano interfacce che potrebbero non essere usate per nulla dagli utenti finali del sistema (interfacce di test) o interfacce che possono essere usate in un contesto differente rispetto al funzionamento reale. D'altro canto, è molto facile e poco costoso eseguire test automatizzati tramite interfacce (API). Testare il SUT tramite di interfacce di test può essere un approccio robusto se sono compresi i rischi potenziali.

Un alto livello di intrusione può portare ad anomalie durante il testing che non si manifestano nelle condizioni d'uso del mondo reale. Se ciò causa esiti negativi con i test automatizzati, la fiducia nella soluzione di automazione del testing può scendere drammaticamente. Gli sviluppatori possono richiedere che gli esiti negativi identificati dal testing automatizzato debbano essere riprodotti manualmente, se possibile, per essere aiutati nell'analisi.

© International Software Testing Qualifications Board

Testing dei componenti del framework
Come ogni altro progetto di sviluppo software, i componenti del framework automatizzato devono essere testati e verificati individualmente. Questo può includere testing funzionale e non-funzionale (prestazioni, utilizzo delle risorse, usabilità, ecc.).

Per esempio, i componenti che forniscono controlli sugli oggetti sulle GUI dei sistemi devono essere testati per un ampio spettro di classi di oggetti, per stabilire che il controllo degli oggetti funzioni correttamente. Ugualmente, log degli errori e report devono produrre informazioni accurate sullo stato dell'automazione e sul comportamento del SUT.

Esempi di testing non-funzionale possono includere la comprensione del degrado delle prestazioni del framework, l'utilizzo delle risorse che può indicare problemi come i memory leak, l'interoperabilità dei componenti dentro e/o fuori il framework.

7.2 Verificare la Suite di Test Automatizzati

Le suite di test automatizzati devono essere testate per completezza, consistenza e comportamento corretto. Possono essere applicati diversi tipi di controlli per assicurare che la suite di test automatizzata sia funzionante in ogni dato momento o per determinare che sia pronta per l'utilizzo.

Ci sono alcuni passi che possono essere intrapresi per provare la suite di test automatizzata. Questi includono:

- Eseguire script di test con esiti di esecuzione positivi (pass) e negativi (fail) noti
- Controllare la suite di test
- Verificare nuovi test incentrati su nuove funzionalità del framework
- Considerare la ripetibilità dei test
- Controllare che siano presenti sufficienti punti di verifica nella suite dei test automatizzati

Ciascuno di questi punti è spiegato in maggiore dettaglio di seguito.

Eseguire script di test con esiti di esecuzione positivi (pass) e negativi (fail) noti
Quando casi di test per cui è noto un esito positivo di esecuzione (pass) falliscono, è immediatamente chiaro che qualche cosa di fondamentale è sbagliato e deve essere corretto quanto prima. Viceversa, quando una suite di test passa mentre sarebbe dovuta fallire, è necessario identificare il caso di test che non ha funzionato correttamente. È importante controllare la corretta generazione dei file di log e delle metriche delle prestazioni, come pure setup e teardown automatizzati dei casi/script di test. È anche utile eseguire alcuni test dei differenti tipi e livelli di test (test funzionali, test prestazionali, testi di componente, ecc.).

Controllare la suite di test
Verificare la suite di test per completezza (tutti i casi di test hanno i risultati attesi, i dati di test sono presenti) e corretta versione rispetto al framework e al SUT.

Verificare nuovi test incentrati su nuove funzionalità del framework
La prima volta che una nuova funzionalità della TAS è usata effettivamente nei casi di test, deve essere verificata e monitorata attentamente per assicurare che stia funzionando correttamente.

© International Software Testing Qualifications Board

Considerare la ripetibilità dei test
Quando si ripetono i test, il risultato/esito del test deve essere sempre lo stesso. Se si hanno dei casi di test nell'insieme dei test che non danno un risultato affidabile (per es. il caso di race condition) può essere opportuno rimuoverli dalla suite di test automatizzati attiva e analizzarli separatamente per trovare la causa originale. Altrimenti si spenderà ripetutamente del tempo ad ogni esecuzione per analizzare il problema.

Gli esiti negativi intermittenti devono essere analizzati. Il problema può risiedere nel caso di test stesso o nel framework (o potrebbe esserci anche un problema nel SUT). L'analisi dei file di log (del caso di test, del framework e del SUT) può identificare la causa originale del problema. Può anche essere necessario effettuare il debugging degli script. Per trovare la causa originale può essere necessario il supporto delle figure di analista del test, di sviluppatore software e di esperto del dominio.

Controllare che siano presenti sufficienti punti di verifica nella suite dei test automatizzati
Deve essere possibile verificare che la suite di test automatizzati sia stata eseguita e abbia ottenuto i risultati attesi, Devono essere fornite delle evidenze per dimostrare che la suite di test e/o i casi di test siano stati eseguiti come atteso. Queste evidenze possono comprendere il logging all'inizio e alla fine di ciascun caso di test, la registrazione dello stato di esecuzione del testing per ogni caso di test completato, la verifica che le post-condizioni sono state ottenute, ecc.

© International Software Testing Qualifications Board

8. Miglioramento Continuo - 150 min.

Termini
manutenzione

Obiettivi di apprendimento per il Miglioramento Continuo

8.1 Opzioni per Migliorare l'Automazione del Testing

ALTA-E-8.1.1 (K4) Analizzare gli aspetti tecnici di una soluzione di automazione del testing rilasciata e fornire le raccomandazioni per il suo miglioramento

8.2 Pianificare l'Implementazione dei Miglioramenti per l'Automazione del Testing

ALTA-E-8.2.1 (K4) Analizzare il testware automatizzato, inclusi i componenti dell'ambiente di test, gli strumenti e le librerie di funzioni a supporto, per capire dove dovrebbero essere fatti consolidamenti e aggiornamenti a seguito di un dato insieme di cambiamenti nell'ambiente di test o del SUT

© International Software Testing Qualifications Board

8.1 Opzioni per Migliorare l'Automazione del Testing

In aggiunta alle continue attività di manutenzione necessarie per manutenere la TAS sincronizzata con il SUT, ci sono tipicamente molte opportunità per migliorare la TAS. Il miglioramento della TAS può essere intrapreso per ottenere una varietà di benefici come maggiore efficienza (riducendo ulteriormente l'intervento manuale), maggiore facilità d'uso, funzionalità aggiuntive e migliore supporto alle attività di testing. La decisione di come migliorare la TAS sarà influenzata dai benefici che aggiungeranno maggior valore al progetto.

Aree specifiche di una TAS che possono essere prese in considerazione per il miglioramento includono scripting, verifiche, architettura, pre- e post-processing, documentazione e supporto dello strumento. Tutte queste aree sono descritte in maggior dettaglio di seguito.

Scripting
Gli approcci di scripting variano dal semplice approccio strutturato agli approcci data-driven fino ai più sofisticati approcci keyword-driven, come descritto nel paragrafo 3.2.2. Può essere opportuno aggiornare l'approccio attuale di scripting della TAS per tutti i nuovi test automatizzati. L'approccio può essere aggiornato per tutti i test automatizzati esistenti o almeno per quelli che richiedono il più alto sforzo di manutenzione.

Piuttosto che cambiare del tutto l'approccio di scripting, il miglioramento della TAS può concentrarsi sull'implementazione degli script. Per esempio:

- Valutare la sovrapposizione di casi di test/passi/procedure nel tentativo di consolidare i test automatizzati.
  I casi di test che contengono sequenze di azioni simili non dovrebbero implementare tali passi più volte. Questi passi devono essere realizzati con una funzione e aggiunti ad una libreria, in modo tale che possano essere riutilizzati. Queste funzioni di libreria possono esser usate da casi di test differenti. Ciò incrementa la manutenibilità del testware. Quando i passi di test non sono identici ma simili, può essere necessaria una parametrizzazione.
  Nota: questo è un tipico approccio nel testing keyword-driven.
- Stabilire un processo di recovery degli errori per la TAS e il SUT.
  Quando si presenta un errore durante l'esecuzione dei casi di test, il TAS deve essere in grado di riprendere a funzionare dalla condizione di errore per poter continuare con il caso di test successivo. Quando avviene un errore nel SUT, la TAS deve essere in grado di eseguire le necessarie azioni di ripristino sul SUT (per es. un reboot di tutto il SUT).
- Valutare le procedure di attesa per assicurare si stia usando il migliore.
  Ci sono 3 comuni procedure di attesa:
  1. Attese cablate nel codice (si attende un certo numero di millisecondi), possono essere causa di molti problemi per l'automazione del testing.
  2. Attesa dinamica tramite polling, per esempio si attende un certo cambiamento di stato o che un'azione sia effettuata, è molto più flessibile ed efficiente:
     - Si attende solo il tempo necessario e non si perde tempo di test
     - Quando per qualche ragione il processo impiega un tempo maggiore, la procedura di polling aspetterà fino a quando la condizione non sarà vera. È necessario ricordarsi di includere un meccanismo di time-out, altrimenti il test può aspettare per sempre in caso di problemi.

© International Software Testing Qualifications Board

3. Un modo ancora migliore è tramite sottoscrizione al gestore di eventi del SUT. Questa è opzione molto più affidabile delle due precedenti, ma il linguaggio di test scripting deve supportare la sottoscrizione agli eventi del SUT e quest'ultimo deve offrire tali eventi all'applicazione di test. È necessario ricordarsi di includere un meccanismo di time-out, altrimenti il test può aspettare per sempre in caso di problemi.

- Trattare il testware come software.
  Sviluppo e manutenzione del testware sono solo una forma di sviluppo di software. Come tale devono essere applicate buone pratiche di codifica (per es. usare linee guida di codifica, analisi statica, revisioni del codice). Può anche essere una buona idea usare sviluppatori software (invece di test engineer) per sviluppare certe parti del testware (per es. librerie).
- Valutare gli script esistenti ai fini della loro revisione/eliminazione.
  Diversi script possono essere problematici (per es. falliscono ogni tanto o hanno alti costi di manutenzione), e può essere saggio riprogettare tali script. Altri script di test possono essere rimossi dalla suite perché non aggiungono più alcun valore.

Esecuzione dei test
Quando una suite di test di regressione automatizzata non termina l'esecuzione durante la notte, non dove essere una sorpresa. Quando il testing richiede troppo tempo di esecuzione, può essere necessario testare in parallelo su diversi sistemi, ma ciò non è sempre possibile. Quando sono usati sistemi (target) costosi per il testing, si può avere il vincolo di dover eseguire tutti i test su un singolo sistema target. Può essere necessario dividere la suite di test di regressione in più parti, ciascuna delle quali viene eseguita in un intervallo di tempo definito (per es. durante una notte). Ulteriori analisi della copertura del test automatizzato possono rivelare delle duplicazioni. Rimuovere le duplicazioni può ridurre il tempo di esecuzione e permettere di ottenere ulteriori efficienze.

Verifiche
Prima di creare nuove funzioni di verifica, si deve adottare un insieme di metodi standard di verifica da usare per tutti i test automatizzati. Questo eviterà di implementare di nuovo azioni di verifica per molti test. Quando i metodi di verifica non sono identici ma simili, l'uso della parametrizzazione aiuterà a consentire l'uso di una funzione attraverso molti tipi di oggetti.

Architettura
Può essere necessario cambiare l'architettura per supportare i miglioramenti della testabilità del SUT. Questi cambiamenti possono essere apportati all'architettura del SUT e/o all'architettura dell'automazione. Questo può costituire un miglioramento importante dell'automazione del testing, ma può richiedere cambiamenti e investimenti significativi nel SUT/TAS. Per esempio, se il SUT viene cambiato per prevedere delle API per il testing, allora anche la TAS deve essere corrispondentemente rivista. Aggiungere questo tipo di funzionalità nelle fasi avanzate può essere piuttosto costoso; è molto meglio pensarci all'inizio dell'automazione (e nelle prime fasi di sviluppo del SUT – si veda il paragrafo 2.3 Progettare per la Testabilità e l'Automazione).

Pre- e post-processing
Fornire attività standard di setup e teardown. Queste sono anche conosciute come pre-processing (setup) e post-processing (teardown). Ciò evita di dover implementare ripetutamente tali attività per ciascun test automatizzato, riducendo non solo i costi di manutenzione ma anche lo sforzo richiesto per implementare nuovi test automatizzati.

© International Software Testing Qualifications Board

Documentazione
Questa copre tutte le forme di documentazione dalla documentazione degli script (cosa fanno gli script, come devono essere usati, ecc.), alla documentazione utente per la TAS, ai report e log prodotti dalla TAS.

Funzionalità della TAS
Aggiungere caratteristiche e funzioni alla TAS come reportistica di dettaglio, log, integrazione con altri sistemi, ecc. Si devono aggiungere nuove funzioni solo quando saranno usate veramente. Aggiungere funzioni non usate incrementa solo la complessità e diminuisce affidabilità e manutenibilità.

Aggiornamenti della TAS
Aggiornando o allineando la TAS a nuove versioni, possono diventare disponibili nuove funzioni che possono essere usate dai casi di test (o possono essere corretti dei malfunzionamenti). Il rischio è che aggiornando il framework (aggiornando gli strumenti di test esistenti o introducendone di nuovi) si potrebbe avere un impatto negativo sui casi di test esistenti. Si deve testare la versione dello strumento di test eseguendo test scelti a campione prima di effettuare il rollout della nuova versione. Tali test scelti a campione devono essere rappresentativi dei test automatizzati di diverse applicazioni, dei diversi tipi di test e, ove appropriato, dei diversi ambienti.

8.2 Pianificare l'Implementazione dei Miglioramenti per l'Automazione del Testing

I cambiamenti ad una TAS esistente richiedono accurata pianificazione e analisi. Si impiega molto sforzo nel creare una TAS robusta che comprende una TAF e librerie di componenti. Ogni cambiamento, non importa quanto banale, può avere un impatto ad ampio spettro su affidabilità e prestazioni della TAS.

Identificare i cambiamenti nei componenti dell'ambiente di test
Si deve valutare quali cambiamenti e miglioramenti devono essere fatti. Questi richiedono cambiamenti al software di test, alle librerie di funzioni personalizzate, al sistema operativo? Ciascuno di questi ha un impatto su come la TAS lavora? L'obiettivo generale è assicurare che i test automatizzati continuino ad essere eseguiti in modo efficiente. I cambiamenti devono essere effettuati incrementalmente in modo che l'impatto sulla TAS possa esser misurato con un numero limitato di esecuzioni di script di test. Una volta che si sia verificata l'assenza di effetti negativi, le modifiche possono essere completamente implementate. L'esecuzione di una regressione completa è il passo finale per validare che le modifiche non abbiano influenzato negativamente gli script automatizzati. Durante l'esecuzione di questi script di regressione, possono essere trovati degli errori. Identificare la causa di questi errori (attraverso report, log, analisi dei dati, ecc.) sarà il mezzo per assicurare che tali errori non siano il risultato dell'attività di miglioramento dell'automazione.

Incrementare efficienza ed efficacia delle librerie delle funzioni principali della TAS
Come una TAS matura, sono scoperti nuovi modi per svolgere compiti in modo più efficiente. Queste nuove tecniche (che includono l'ottimizzazione del codice delle funzioni, l'uso di librerie di sistema operativo più recenti, ecc.) devono essere incorporate nelle librerie delle funzioni principali che sono usate dal progetto corrente e da tutti gli altri progetti.

© International Software Testing Qualifications Board

Individuare le varie funzioni che agiscono sullo stesso tipo di controllo per razionalizzarle
Una larga parte di quello che viene svolto durante l'esecuzione di test automatizzati è l'interrogazione di controlli della GUI. Queste interrogazioni servono a fornire informazioni su tali controlli (per es. visibile/ non visibile, attivo/non attivo, dimensioni, dati, ecc.). Con tali informazioni un test automatizzato può selezionare una voce da un menu a discesa, inserire dati in un campo, leggere un valore da un campo, ecc. Ci sono diverse funzioni che possono agire sui controlli per ottenere tali informazioni. Alcune funzioni sono estremamente specializzate, mentre altre sono più generali. Per esempio, ci può essere una funzione specifica che lavora solo su menu a discesa. In alternativa, ci può essere una funzione (o può essere creata ed usata all'interno della TAS) che lavora con diverse funzioni specificando la funzione specifica tra uno dei suoi parametri. Quindi un TAE può usare diverse funzioni che possono essere consolidate in un numero minore di funzioni, ottenendo gli stessi risultati e minimizzando lo sforzo di manutenzione necessario.

Modificare la TAA per adattarla ai cambiamenti del SUT
Durante la vita di una TAS, devono essere fatte delle modifiche per adattarla ai cambiamenti che subisce il SUT. Cosi come il SUT evolve e matura, anche la TAA sottostante dovrà evolvere per continuare ad assicurare supporto per il SUT. Deve essere posta cura quando si estendono le funzionalità in modo che la loro implementazione non avvenga in modo affrettato e che invece tali funzionalità siano analizzate e cambiate a livello di architettura della soluzione automatizzata. Questo assicurerà che quando nuove funzionalità del SUT richiederanno script aggiuntivi, componenti compatibili saranno già pronti per favorire la preparazione di questi nuovi test automatizzati.

Convenzioni sui nomi e standardizzazione
Quando sono introdotti dei cambiamenti, le convenzioni sui nomi per il nuovo codice di automazione e per le librerie di funzioni devono essere consistenti con gli standard definiti in precedenza (si veda il paragrafo 4.3.2 Ambito e Approccio).

Valutazione degli script esistenti per un SUT ai fini della loro revisione/eliminazione
Il processo di cambiamento e miglioramento include anche una valutazione degli script esistenti, del loro utilizzo e del loro persistente valore. Per esempio, se certi test sono complessi e impiegano molto tempo per l'esecuzione, può essere più sostenibile ed efficiente decomporli in più test più piccoli. Individuare ed eliminare i test che sono eseguiti di rado o per niente snellirà la complessità della TAS e porterà maggiore chiarezza su quello che deve essere manutenuto.

© International Software Testing Qualifications Board

9. Riferimenti

9.1 Standard

Tra gli standard per l'automazione del testing sono compresi (ma non solo):

- La Testing and Test Control Notation (TTCN-3) di ETSI (European Telecommunication Standards Institute) e ITU (International Telecommunication Union) che consiste in
  - 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
- L'Automatic Test Markup Language (ATML) dello IEEE (Institute of Electrical and Electronics Engineers) che consiste di:
  - 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
- Lo Standard ISO/IEC/IEEE 29119-3
- Lo UML Testing Profile (UTP) dell'OMG (Object Management Group) che specifica i concetti di specifica del test:
  - Architettura del Testing
  - Dati di test
  - Comportamento del test
  - Test logging
  - Gestione del testing

© International Software Testing Qualifications Board

9.2 Documenti ISTQB

Identificativo	Riferimento
ISTQB-AL-TM	ISTQB Certified Tester, Advanced Level Syllabus, Test Manager, Version 2012, disponibile sul sito [ISTQB-Web]
ISTQB-AL-TTA	ISTQB Certified Tester, Advanced Level Syllabus, Technical Test Analyst, Version 2012, disponibile sul sito [ISTQB-Web]
ISTQB-EL-CEP	ISTQB Advanced Level Certification Extension, available from [ISTQB-Web]
ISTQB-EL-Modules	ISTQB Advanced Level Modules Overview, Version 1.2, August 23, 2013, disponibile sul sito [ISTQB-Web]
ISTQB-EL-TM	ISTQB Advanced Level – Test Management Syllabus, Version 2011, disponibile sul sito [ISTQB-Web]
ISTQB-FL	ISTQB Foundation Level syllabus, Version 2011, available from [ISTQB-Web]
ISTQB-Glossary	ISTQB Glossary of terms, Version 2.4, July 4, 2014, disponibile sul sito [ISTQB-Web]

9.3 Marchi registrati

In questo documento sono utilizzati i seguenti marchi registrati e marchi di servizio:

ISTQB® è un marchio registrato dell'International Software Testing Qualifications Board

9.4 Libri

Identificativo	Riferimento
[Baker08]	Paul Baker, Zhen Ru Dai, Jens Grabowski and Ina Schieferdecker, "Model-Driven Testing: Using the UML Testing Profile", Springer 2008 edition, ISBN-10: 3540725628, ISBN-13: 978-3540725626
[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

© International Software Testing Qualifications Board

[Fewster&Graham12] Mark Fewster, Dorothy Graham, "Experiences of Test Automation: Case Studies of Software Test Automation", Addison-Wesley, 2012

[Fewster&Graham99] Mark Fewster, Dorothy Graham, "Software Test Automation: Effective use of test execution tools", ACM Press Books, 1999, ISBN-10: 0201331403, ISBN-13: 9780201331400

[McCaffrey06] James D. McCaffrey, ".NET Test Automation Recipes: A Problem-Solution Approach", APRESS, 2006 ISBN-13:978-1-59059-663-3, ISBN-10:1-59059-663-3

[Mosley02] Daniel J. Mosley, Bruce A. Posey, "Just Enough Software Test Automation", Prentice Hall, 2002, ISBN-10: 0130084689, ISBN-13: 9780130084682

[Willcock11] Colin Willcock, Thomas Deiß, Stephan Tobies and Stefan Keil, "An Introduction to TTCN-3" Wiley, 2nd edition 2011, ISBN-10: 0470663065, ISBN-13: 978-0470663066

9.5 Riferimenti Web

Identificativo	**Riferimento**
ISTQB-Web	Sito Web dell'International Software Testing Qualifications Board. Si faccia riferimento a questo sito per le ultime versioni dei Glossari e dei Syllabi ISTQB.

© International Software Testing Qualifications Board

10. Nota per i Training Providers

10.1Tempi di apprendimento

Per ogni capitolo in questo Syllabus è assegnato un tempo in minuti. Lo scopo è sia di dare una guida sulle relative proporzioni di tempo che devono essere considerate per ciascuna sessione di un corso accreditato e sia di dare un tempo minimo approssimativo di insegnamento di ogni sessione

I Training provider potranno impiegare più tempo di quanto indicato e i candidati possono impiegare più tempo ancora nella lettura e nella ricerca. Il piano di un corso non deve seguire lo stesso ordine del Syllabus. Non è richiesto di condurre il corso in blocco di tempo continuo.

La tabella seguente fornisce una linea guida per i tempi di insegnamento e di esercitazione per ciascun capitolo (tutti i tempi sono indicati in minuti).

Capitolo	**Minuti**
0. Introduzione	0
1. Introduzione e obiettivi per l'automazione del testing	15
2. Preparazione per l'Automazione del Testing	180
3. Architettura Generica di automazione dei testing	300
4. Rischi e imprevisti dell'utilizzo di una TAS	315
5. Report e metriche per l'Automazione del testing	195
6. Transizione dal testing manuale ad un ambiente automatizzato	210
7. Verificare la TAS	120
8. Miglioramento continuo	240
Totale:	1575

Il tempo totale del corso in giorni, basato sulla media di sette ore per giorno lavorativo, è: 2 giorni, 5 ore, 30 minuti.

10.2Esercizi

Non ci sono esercizi definiti che possono essere eseguiti.

10.3Regole per l'e-Learning

Tutte le parti di questo Syllabus sono considerate appropriate per essere implementate via e-learning.

© International Software Testing Qualifications Board