

Certified Tester Advanced Level Test Automation Engineering Syllabus

Versione 2.0

International Software Testing Qualifications Board



Avviso di copyright

Avviso di copyright © International Software Testing Qualifications Board (di seguito denominato ISTQB®)
ISTQB® è un marchio registrato di International Software Testing Qualifications Board.

Copyright © 2024, gli autori del Test Automation Engineering v2.0 syllabus: Andrew Pollner (presidente), Péter Földházi, Patrick Quilter, Gergely Ágnes, László Szikszai

Copyright © 2016, gli autori Andrew Pollner (presidente), Bryan Bakker, Armin Born, Mark Fewster, Jani Haukinen, Raluca Popescu, Ina Schieferdecker.

Tutti i diritti riservati. Con la presente gli autori trasferiscono il copyright a ISTQB®. Gli autori (come attuali titolari del copyright) e ISTQB® (come futuro titolare del copyright) hanno concordato le seguenti condizioni di utilizzo:

È possibile copiare estratti di questo documento per uso non-commerciale, a condizione che venga riconosciuta la fonte. Qualsiasi Training Provider accreditato può utilizzare il presente syllabus come base per un corso di formazione, a condizione che gli autori e ISTQB® siano riconosciuti come fonte e proprietari del copyright del syllabus e a condizione che qualsiasi pubblicità di tale corso di formazione possa menzionare il syllabus solo dopo l'accREDITAMENTO ufficiale dei materiali di formazione da parte di un Member Board riconosciuto da ISTQB®.

Qualsiasi individuo o gruppo di individui può utilizzare questo syllabus come base per articoli e libri, a condizione che gli autori e ISTQB® siano riconosciuti come fonte e proprietari del copyright del syllabus.

È proibito qualsiasi altro utilizzo di questo syllabus senza aver prima ottenuto l'approvazione scritta di ISTQB®.

Qualsiasi Member Board riconosciuto da ISTQB® può tradurre questo syllabus a condizione che riproduca il suddetto avviso di copyright nella versione tradotta del syllabus.

Storico delle review

Versione	Data	Osservazioni
Syllabus 2016	2016/10/21	Rilascio CTAL-TAE GA
Syllabus v2.0	2024/05/03	Rilascio CTAL-TAE v2.0 GA

Indice dei contenuti

Avviso di copyright	2
Storico delle review	3
Indice dei contenuti	4
Ringraziamenti	7
0 Introduzione	8
0.1 Scopo di questo Syllabus	8
0.2 Test Automation Engineering nel Testing del Software	8
0.3 Percorso di Carriera per i Tester e i Test Automation Engineer	9
0.4 Business Outcome	9
0.5 Obiettivi di Apprendimento Esaminabili e Livelli di Conoscenza	10
0.6 L'Esame per il Certificato Test Automation Engineering	10
0.7 Accredimento	11
0.8 Gestione degli Standard	11
0.9 Mantenere l'Attualità	11
0.10 Livello di dettaglio	11
0.11 Come è Organizzato questo Syllabus	12
1 Introduzione e Obiettivi della Test Automation - 45 minuti (K2)	14
1.1 Scopo della Test Automation	15
1.1.1 Spiegare i vantaggi e gli svantaggi della test automation	15
1.2 La Test Automation nel Ciclo di Vita dello Sviluppo Software	16
1.2.1 Spiegare come la test automation si applica a diversi modelli del ciclo di vita dello sviluppo software	16
1.2.2 Selezionare gli strumenti di test automation adatti per un determinato system under test	16
2 Preparazione per la Test Automation - 180 minuti (K4)	18
2.1 Comprendere la Configurazione di un'Infrastruttura per Consentire la Test Automation	19
2.1.1 Descrivere le esigenze di configurazione di un'infrastruttura che consente l'implementazione della test automation	19
2.1.2 Spiegare come la test automation viene sfruttata in ambienti diversi	19
2.2 Processo di Valutazione per Selezionare gli Strumenti e le Strategie Adatti	21
2.2.1 Analizzare un system under test per determinare la test automation solution appropriata	21
2.2.2 Illustrare i risultati tecnici della valutazione di uno strumento	21
3 Test Automation Architecture - 210 minuti (K3)	23

3.1	Concetti di Progettazione Adottati nella Test Automation	24
3.1.1	Spiegare le caratteristiche principali di una test automation architecture	24
3.1.2	Spiegare come progettare una test automation solution	25
3.1.3	Applicare il layering nei test automation framework	25
3.1.4	Applicare i diversi approcci per automatizzare i test case	26
3.1.5	Applicare principi di progettazione e modelli di progettazione nella test automation	29
4	Implementazione della Test Automation - 150 minuti (K4)	30
4.1	Sviluppo della Test Automation	31
4.1.1	Applicare linee guida che supportano lo svolgimento efficace di attività pilota e di deployment della test automation	31
4.2	Rischi Associati allo Sviluppo della Test Automation	31
4.2.1	Analizzare i rischi di deployment e pianificare strategie di mitigazione per la test automation	31
4.3	Manutenibilità della Test Automation Solution	33
4.3.1	Spiegare quali fattori supportano e influenzano la manutenibilità di una test automation solution	33
5	Strategie di Implementazione e Deployment per la Test Automation - 90 minuti (K3)	35
5.1	Integrazione con le Pipeline di CI/CD	36
5.1.1	Applicare la test automation a diversi livelli di test all'interno delle pipeline	36
5.1.2	Spiegare il configuration management del testware	37
5.1.3	Spiegare le dipendenze della test automation per un'infrastruttura API	38
6	Reporting e Metriche della Test Automation - 150 minuti (K4)	39
6.1	Raccolta, Analisi e Reporting dei Dati della Test Automation	40
6.1.1	Applicare i metodi di raccolta dei dati dalla test automation solution e dal system under test	40
6.1.2	Analizzare i dati dalla test automation solution e dal system under test per comprendere meglio i risultati dei test	42
6.1.3	Spiegare come un test progress report viene costruito e pubblicato	43
7	Verifica della Test Automation Solution - 135 minuti (K3)	45
7.1	Verifica dell'Infrastruttura di Test Automation	46
7.1.1	Pianificare la verifica dell'ambiente di test automation, compresa la configurazione degli strumenti di test	46
7.1.2	Spiegare il comportamento corretto di un determinato test script e/o test suite automatizzata	47
7.1.3	Identificare dove la test automation produce risultati inattesi	48

7.1.4	Spiegare in che modo l'analisi statica può aiutare a migliorare la qualità del codice della test automation	48
8	Miglioramento Continuo - 210 minuti (K4)	50
8.1	Opportunità di Miglioramento Continuo per la Test Automation.....	51
8.1.1	Scoprire opportunità per migliorare i test case attraverso la raccolta e l'analisi dei dati	51
8.1.2	Analizzare gli aspetti tecnici di una test automation solution distribuita e fornire raccomandazioni per il miglioramento.....	51
8.1.3	Ristrutturare il testware automatizzato per allinearli agli aggiornamenti del SUT	54
8.1.4	Riassumere le opportunità per l'utilizzo degli strumenti di test automation	55
9	Riferimenti	57
10	Appendice A - Obiettivi di Apprendimento/Livello Cognitivo di Conoscenza	60
11	Appendice B - Matrice di Tracciabilità dei Business Outcome rispetto agli Obiettivi di Apprendimento	63
12	Appendice C – Release Note	67
13	Appendice D - Termini Specifici del Dominio	68
14	Indice	69

Ringraziamenti

Questo documento è stato formalmente rilasciato dalla General Assembly di ISTQB® il 3 maggio 2024.

È stato prodotto dalla Test Automation Task Force dello Specialist Working Group di International Software Testing Qualifications Board: Graham Bath (presidente dello Specialist Working Group), Andrew Pollner (vicepresidente dello Specialist Working Group e presidente della Test Automation Task Force), Péter Földhási, Patrick Quilter, Gergely Ágnes, László Szikszai. I revisori della Test Automation Task Force includono: Armin Beer, Armin Born, Geza Bujdosó, Renzo Cerquozzi, Jan Giesen, Arnika Hryszko, Kari Kakkonen, Gary Mogyorodi, Chris van Bael, Carsten Weise, Marc-Florian Wendland.

Reviewer tecnico: Gary Mogyorodi

Le seguenti persone hanno partecipato alla revisione, ai commenti e alla votazione di questo syllabus:

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

ISTQB Working Group Advanced Level Test Automation Engineer (Edizione 2016): Andrew Pollner (presidente), Bryan Bakker, Armin Born, Mark Fewster, Jani Haukinen, Raluca Popescu, Ina Schieferdecker.

Traduzione italiana v2.0 a cura di ITA-STQB (www.ita-stqb.org) – info@ita-stqb.org

Revisione della traduzione italiana v2.0 a cura di: Raffaele Bozzini, Alessandro Collino, Salvatore Reale.

0 Introduzione

0.1 Scopo di questo Syllabus

Questo syllabus costituisce la base per l'International Software Testing Qualification per l'Advanced Level Test Automation Engineering (CTAL-TAE). ISTQB® fornisce questo syllabus come segue:

1. Ai Member Board, per la traduzione nella loro lingua locale e per l'accreditamento dei Training Provider. I Member Board possono adattare il syllabus alle loro specifiche esigenze linguistiche e modificare i riferimenti per adattarli alle loro pubblicazioni locali.
2. Agli organismi di certificazione, per derivare le domande d'esame nella loro lingua locale adatte agli obiettivi di apprendimento di questo syllabus.
3. Ai Training Provider, per produrre materiale didattico e determinare i metodi di insegnamento appropriati.
4. Ai candidati alla certificazione, per prepararsi all'esame di certificazione (come parte di un corso di formazione o in modo indipendente).
5. Alla comunità internazionale di System Engineering e Software Engineering, per promuovere la professione del testing del software e dei sistemi e come base per libri e articoli.

0.2 Test Automation Engineering nel Testing del Software

La certificazione di Test Automation Engineering è rivolta a chiunque sia coinvolto nel testing del software e nella test automation. Ciò include persone che ricoprono ruoli come tester, test analyst, test automation engineer, consulenti di testing, architetti di testing, test manager e sviluppatori di software. Questa certificazione è adatta anche a tutti coloro che desiderano una comprensione di base della test automation, come project manager, quality manager, software development manager, business analyst, IT director e consulenti di gestione.

Il Test Automation Engineering syllabus è rivolto ai test engineer che desiderano implementare o migliorare la test automation. Definisce i metodi e le pratiche che possono supportare una soluzione sostenibile.

Altre linee guida e modelli di riferimento relativi alle soluzioni di test automation sono gli standard di ingegneria del software per i cicli di vita dello sviluppo del software selezionati, le tecnologie di programmazione e gli standard di formattazione. Questo syllabus non insegna l'ingegneria del software. Tuttavia, ci si aspetta che un test automation engineer abbia capacità, esperienza e competenza nell'ingegneria del software.

Inoltre, un test automation engineer deve conoscere gli standard di programmazione e di documentazione del settore e le best practice per utilizzarli durante lo sviluppo di una test automation solution. Queste pratiche possono aumentare la manutenibilità, l'affidabilità e la sicurezza della test automation solution. Tali standard sono tipicamente basati su caratteristiche di qualità.

0.3 Percorso di Carriera per i Tester e i Test Automation Engineer

Lo schema ISTQB® fornisce supporto ai professionisti del testing in tutte le fasi della loro carriera, offrendo una conoscenza ampia e approfondita. Chi ha conseguito la certificazione ISTQB® Test Automation Engineering può essere interessato anche alla certificazione Test Automation Strategy (CT-TAS).

Chi ha conseguito la certificazione ISTQB® Certified Tester Test Automation Engineering può essere interessato anche alle altre certificazioni Core Advanced Level (Test Analyst, Technical Test Analyst e Test Manager) e successivamente alle certificazioni Expert Level (Test Management o Improving the Test Process). Chiunque voglia sviluppare competenze nelle pratiche del testing in un ambiente Agile può prendere in considerazione le certificazioni Agile Technical Tester o Agile Test Leadership at Scale. Il percorso di certificazione Specialist offre un approfondimento in aree che hanno specifici approcci di test e attività di test, ad esempio Test Automation Strategy, Performance Testing, Security Testing, AI Testing e Mobile Application Testing, o dove è richiesta una conoscenza specifica del dominio (ad esempio, Automotive Software Testing o Game Testing). È possibile visitare il sito www.istqb.org per le ultime informazioni sul Certified Tester Scheme di ISTQB®.

0.4 Business Outcome

Questo paragrafo elenca i Business Outcome (Risultati di Business) previsti da una persona che ha conseguito la certificazione Test Automation Engineering.

Un tester che ha conseguito la certificazione Test Automation Engineering è in grado di...

TAE-B01	Descrivere lo scopo della test automation
TAE-B02	Comprendere la test automation attraverso il ciclo di vita dello sviluppo software
TAE-B03	Comprendere la configurazione di un'infrastruttura per abilitare la test automation
TAE-B04	Imparare il processo di valutazione per selezionare gli strumenti e le strategie adeguate
TAE-B05	Comprendere i concetti di progettazione per la costruzione di soluzioni di test automation modulari e scalabili.
TAE-B06	Selezionare un approccio, compreso un pilota, per pianificare la test automation all'interno del ciclo di vita dello sviluppo software
TAE-B07	Progettare e sviluppare soluzioni di test automation (nuove o modificate) che soddisfino le esigenze tecniche
TAE-B08	Considerare l'ambito e l'approccio della test automation e la manutenzione del testware
TAE-B09	Comprendere come i test automatizzati si integrano nelle pipeline di CI/CD
TAE-B10	Comprendere come raccogliere, analizzare e fornire report sui dati della test automation per informare gli stakeholder
TAE-B11	Verificare l'infrastruttura della test automation
TAE-B12	Definire opportunità di miglioramento continuo per la test automation

0.5 Obiettivi di Apprendimento Esaminabili e Livelli di Conoscenza

Gli Obiettivi di Apprendimento supportano i Business Outcome e vengono utilizzati per creare gli esami di Certified Tester Test Automation Engineering.

In generale, tutti i contenuti di questo syllabus sono esaminabili ai livelli K2, K3 e K4, ad eccezione di Introduzione e Appendici. Al candidato può quindi essere chiesto di riconoscere, ricordare o richiamare una parola chiave o un concetto menzionato in uno qualsiasi degli otto capitoli. I livelli specifici degli obiettivi di apprendimento sono specificati all'inizio di ogni capitolo e sono classificati come segue:

- K2: Capire
- K3: Applicare
- K4: Analizzare

Ulteriori dettagli ed esempi di obiettivi di apprendimento sono riportati nell'Appendice A.

Tutti i termini elencati come parole chiave sotto i titoli dei capitoli devono essere ricordati (K1), anche se non esplicitamente menzionati negli obiettivi di apprendimento.

0.6 L'Esame per il Certificato Test Automation Engineering

L'esame per il Certificato Test Automation Engineering si baserà su questo syllabus. Le risposte alle domande d'esame possono richiedere l'uso di materiale basato su più di un paragrafo di questo syllabus. Tutti i paragrafi del syllabus sono esaminabili, ad eccezione dell'Introduzione e delle Appendici. Gli standard e i libri sono inclusi come riferimenti, ma il loro contenuto non è esaminabile, al di là di quanto sintetizzato in questo syllabus relativamente a tali standard e libri.

Per ulteriori dettagli sull'esame del Certificato Test Automation Engineering, fare riferimento al documento Exam Structures and Rules v1.1 Compatible with Syllabus Foundation and Advanced Levels and Specialists Modules.

Il criterio di accesso per sostenere l'esame di Test Automation Engineering è che i candidati abbiano un interesse per il testing del software e la test automation. Tuttavia, è fortemente consigliato ai candidati anche di:

- Avere almeno un minimo di esperienza nello sviluppo del software e nel testing del software, ad esempio sei mesi di esperienza come ingegnere di test del software o come sviluppatore di software.
- Seguire un corso accreditato secondo gli standard ISTQB (da uno dei member board riconosciuti da ISTQB®).

Nota per i Requisiti di Ingresso: il certificato ISTQB® Foundation Level deve essere ottenuto prima di sostenere l'esame di certificazione ISTQB® Test Automation Engineering.

0.7 Accreditamento

Un Member Board ISTQB® può accreditare i Training Provider, il cui materiale didattico segue questo syllabus. I Training Provider dovrebbero ottenere le linee guida per l'accreditamento dal Member Board o dall'ente che effettua l'accreditamento. Un corso accreditato viene riconosciuto essere conforme a questo syllabus ed è autorizzato a gestire un esame ISTQB® come parte del corso stesso.

Le linee guida per l'accreditamento di questo syllabus seguono le Accreditation Guidelines pubblicate dal Working Group Processes Management and Compliance.

0.8 Gestione degli Standard

Il Test Automation Engineering syllabus contiene riferimenti a standard (ad esempio, IEEE e ISO). Lo scopo di questi riferimenti è di fornire un framework (come nei riferimenti all'ISO 25010 per quanto riguarda le caratteristiche di qualità) o di fornire una fonte di informazioni aggiuntive, se il lettore lo desidera. Si noti che il syllabus utilizza i documenti degli standard come riferimenti. I documenti degli standard non sono oggetto di esame. Per ulteriori informazioni sugli Standard, fare riferimento al capitolo 9 Riferimenti.

0.9 Mantenere l'Attualità

L'industria del software cambia rapidamente. Per far fronte a questi cambiamenti e per fornire agli stakeholder l'accesso a informazioni rilevanti e aggiornate, i gruppi di lavoro ISTQB hanno creato dei link sul sito web www.istqb.org, che rimandano a documenti di supporto e a modifiche agli standard. Queste informazioni non sono oggetto d'esame nell'ambito del Test Automation Engineering syllabus.

0.10 Livello di dettaglio

Il livello di dettaglio di questo syllabus consente di organizzare corsi ed esami consistenti a livello internazionale. Per raggiungere questo obiettivo, il syllabus consiste di:

- Obiettivi didattici generali che descrivono i propositi del Test Automation Engineering Specialist
- Una lista di termini (parole chiave) che gli studenti devono essere in grado di ricordare.
- Obiettivi di apprendimento per ogni area di conoscenza, che descrivono i risultati cognitivi di apprendimento che devono essere raggiunti.
- Una descrizione dei concetti chiave, che include i riferimenti a fonti quali la letteratura o gli standard accettati.

Il contenuto del syllabus non è una descrizione dell'intera area di conoscenza del testing del software, ma riflette il livello di dettaglio che deve essere coperto nei corsi di formazione Test Automation Engineering. Si focalizza sui concetti e sulle tecniche di test che possono essere applicati a tutti i progetti software, compresi quelli che seguono i metodi Agile. Questo syllabus non contiene specifici obiettivi di apprendimento relativi al testing Agile, ma discute come questi concetti si applicano ai progetti Agile e ad altri tipi di progetti.

0.11 Come è Organizzato questo Syllabus

Sono presenti otto capitoli con contenuti esaminabili. Nell'intestazione principale di ogni capitolo è specificato il tempo per il capitolo; la tempistica non viene fornita al di sotto del livello del capitolo. Per i corsi di formazione accreditati, il syllabus richiede un minimo di 21 ore di lezione, distribuite negli otto capitoli come segue:

- Capitolo 1: 45 minuti - Introduzione e Obiettivi della Test Automation
 - Il tester impara i vantaggi della test automation e i suoi limiti
 - Viene coperta la test automation all'interno di diversi modelli del ciclo di vita dello sviluppo software
 - Il tester impara come l'architettura di un system under test (SUT) influisce sull'idoneità degli strumenti di test
- Capitolo 2: 180 minuti - Preparazione per la Test Automation
 - Viene coperta la progettazione per la testabilità del SUT attraverso l'osservabilità, la controllabilità e un'architettura chiaramente definita
 - Il tester apprende la test automation in diversi ambienti
 - Vengono coperti i fattori necessari per valutare una test automation solution appropriata
 - Il tester apprenderà le considerazioni tecniche necessarie per sviluppare raccomandazioni sulla test automation
- Capitolo 3: 210 minuti - Test Automation Architecture
 - Viene coperta la test automation architecture e i suoi componenti che portano a una test automation solution
 - Il tester imparerà i layer e la loro applicazione in un framework di test automation
 - Verranno coperti diversi approcci all'utilizzo di strumenti di test automation
 - Il tester imparerà come i principi di progettazione e i modelli di progettazione possono essere applicati alla test automation
- Capitolo 4: 150 minuti - Implementare la Test Automation
 - Verrà coperto come pianificare e distribuire efficacemente un progetto pilota di test automation
 - Il tester imparerà i rischi del deployment e le strategie di mitigazione
 - Verranno coperti i fattori che migliorano la manutenibilità del codice di test automation
- Capitolo 5: 90 minuti - Strategie di Implementazione e Deployment per la Test Automation
 - Il tester imparerà le pipeline di CI/CD e l'esecuzione dei test automatizzati su tutti i livelli di test
 - Verrà coperto il configuration management per i componenti della test automation
 - Il tester imparerà le dipendenze applicate all'API testing e al contract testing

- Capitolo 6: 150 minuti - Reporting e Metriche della Test Automation
 - Il tester apprenderà dove possono essere raccolti i dati da un SUT e dalla test automation per l'analisi e il reporting
 - Verrà coperta l'analisi dei dati dai report del SUT e dalla test automation per scoprire le cause dei failure
 - Verrà coperto l'utilizzo di test report e dashboard per informare gli stakeholder
- Capitolo 7: 135 minuti - Verifica della Soluzione di Test Automation
 - Il tester imparerà come esaminare e verificare il corretto funzionamento dei componenti di test automation e dell'ambiente di test
 - Verrà coperto il compito di garantire la corretta esecuzione dei test script delle test suite
 - Il tester comprenderà quando eseguire l'analisi delle root cause
 - Verranno coperte le tecniche per analizzare il codice per la qualità della test automation
- Capitolo 8: 210 minuti - Miglioramento Continuo
 - Verranno coperte ulteriori aree di analisi dei dati per il miglioramento dei test case
 - Il tester apprenderà come apportare miglioramenti e aggiornamenti a una test automation solution e ai suoi componenti
 - Verrà coperto come identificare i modi per consolidare e semplificare la test automation
 - Il tester imparerà come gli strumenti di test automation possono assistere con le esigenze del supporto dei test e di configurazione dei test

1 Introduzione e Obiettivi della Test Automation - 45 minuti (K2)

Parole Chiave

system under test, test automation, test automation engineer

Obiettivi di Apprendimento per il Capitolo 1:

1.1 Scopo della Test Automation

TAE-1.1.1 (K2) Spiegare i vantaggi e gli svantaggi della test automation

1.2 La Test Automation nel Ciclo di Vita dello Sviluppo Software

TAE-1.2.1 (K2) Spiegare come la test automation si applica a diversi modelli del ciclo di vita dello sviluppo software

TAE-1.2.2 (K2) Selezionare gli strumenti di test automation adatti per un determinato system under test

1.1 Scopo della Test Automation

1.1.1 Spiegare i vantaggi e gli svantaggi della test automation

La test automation, che comprende l'esecuzione dei test automatizzati e la creazione di test report, consiste in una o più delle seguenti attività:

- Utilizzare strumenti software appositamente creati per controllare e impostare le test suite per l'esecuzione dei test
- Eseguire i test in modo automatizzato
- Confrontare i risultati effettivi con i risultati attesi

La test automation fornisce funzionalità e capacità significative che possono consentire di interagire con un system under test (SUT). La test automation può coprire un'ampia area del software. Le soluzioni coprono molti tipi di software (ad esempio, SUT con interfaccia utente (UI), SUT senza UI, applicazioni mobili, protocolli di rete e connessioni).

La test automation presenta molti vantaggi. Essa:

- Consente l'esecuzione di più test per ogni build rispetto ai test manuali
- Fornisce la possibilità di creare ed eseguire test che non possono essere eseguiti manualmente (ad esempio, risposta in tempo reale, testing remoto e testing parallelo)
- Consente di eseguire test più complessi rispetto ai test manuali
- Esegue i test più rapidamente rispetto ai test manuali
- È meno soggetta all'errore umano
- È più efficace ed efficiente nell'uso delle risorse di test
- Fornisce un feedback più rapido sulla qualità del SUT
- Contribuisce a migliorare l'affidabilità del sistema (ad esempio, disponibilità e recuperabilità)
- Migliora la coerenza dell'esecuzione dei test tra i vari cicli di test

Tuttavia, la test automation presenta potenziali svantaggi, tra cui:

- Il progetto comporterà costi aggiuntivi, in quanto potrebbe essere necessario assumere un test automation engineer (TAE), acquistare nuovo hardware e organizzare la formazione
- Necessità di un investimento iniziale per la creazione di una test automation solution
- Tempo per sviluppare e mantenere una test automation solution
- Necessità di stabilire obiettivi di test automation chiari per garantire il successo delle relative attività
- Rigidità dei test e minore adattabilità ai cambiamenti nel SUT
- Introduzione di ulteriori difetti tramite la test automation

La test automation ha delle limitazioni che devono essere tenute presenti:

- Non tutti i test manuali possono essere automatizzati
- Verifica solo ciò che i test automatizzati sono programmati a eseguire
- La test automation può controllare solo risultati di test interpretabili dalla macchina, il che significa che alcune caratteristiche di qualità potrebbero non essere testabili con l'automazione
- La test automation può controllare solo i risultati di test che possono essere verificati da un oracolo di test automatizzato

1.2 La Test Automation nel Ciclo di Vita dello Sviluppo Software

1.2.1 Spiegare come la test automation si applica a diversi modelli del ciclo di vita dello sviluppo software

Waterfall

Il modello waterfall è un modello di SDLC lineare e sequenziale. Questo modello prevede fasi distinte (requisiti, progettazione, implementazione, verifica e manutenzione), ognuna delle quali si conclude tipicamente con una documentazione che deve essere approvata. L'implementazione della test automation in genere avviene in parallelo o dopo la fase di implementazione. Le esecuzioni dei test di solito avvengono durante la fase di verifica, poiché i componenti del software non sono pronti per il testing fino a quel momento.

V-model

Il V-model è un modello di SDLC in cui un processo viene eseguito in modo sequenziale. Man mano che un progetto viene definito dai requisiti di alto livello a quelli di basso livello, vengono definite le corrispondenti attività di testing e integrazione mirate a validare tali requisiti. Da qui derivano i livelli di test tradizionali: componente, integrazione dei componenti, sistema, integrazione dei sistemi e accettazione, come descritto nella Sezione 2.2 del CTFL. Fornire un framework di test automation (TAF) per ogni livello di test è possibile e consigliato.

Sviluppo software Agile

Nello sviluppo software Agile, le possibilità per la test automation sono innumerevoli. A differenza del modello waterfall o del V-model, nel metodo di sviluppo software Agile, i TAE e i rappresentanti di business possono decidere la roadmap, la tempistica e la consegna dei test pianificati. In questo metodo, esistono best practice come le review del codice, la pair programming e l'esecuzione frequente di test automatizzati. L'eliminazione dei 'silos' (mirata ad assicurare una migliore collaborazione tra sviluppatori, tester e altri stakeholder) consente ai team di coprire tutti i livelli di test con la quantità e la profondità adeguate di test automation, raggiungendo un obiettivo chiamato in-sprint automation. Maggiori dettagli sono disponibili nell'ISTQB CT-TAS Syllabus, Sezione 3.2.

1.2.2 Selezionare gli strumenti di test automation adatti per un determinato system under test

Per identificare gli strumenti di test più adatti a un determinato progetto, occorre innanzitutto analizzare il SUT. I TAE devono identificare i requisiti del progetto che possono essere utilizzati come base per la selezione degli strumenti.

Poiché per il software della UI e, ad esempio, per i servizi web, si utilizzano funzionalità diverse degli strumenti di test automation, è importante comprendere cosa il progetto vuole ottenere nel tempo. Non c'è limite al numero di strumenti di test automation e funzionalità che possono essere utilizzati o selezionati, ma bisogna sempre considerare i costi. L'utilizzo di uno strumento commerciale off-the-shelf o l'implementazione di una soluzione personalizzata basata su tecnologia open-source può essere un processo complesso.

Il prossimo argomento da valutare è la composizione e l'esperienza del team nella test automation. Nel caso in cui i tester abbiano poca o nessuna esperienza di programmazione, l'utilizzo di una soluzione low-code o no-code può essere una scelta valida.

Per i technical tester con conoscenze di programmazione, può essere utile scegliere strumenti il cui linguaggio corrisponda a quello del SUT. Ciò offre vantaggi, tra cui la possibilità di collaborare in modo più efficiente con gli sviluppatori per il debugging dei difetti della test automation e per la formazione trasversale dei membri tra i vari team.

2 Preparazione per la Test Automation - 180 minuti (K4)

Parole Chiave

testing delle API, testing della GUI, testabilità

Obiettivi di Apprendimento per il Capitolo 2:

2.1 Comprendere la Configurazione di un'Infrastruttura per Consentire la Test Automation

TAE-2.1.1 (K2) Descrivere le esigenze di configurazione di un'infrastruttura che consenta l'implementazione della test automation

TAE-2.1.2 (K2) Spiegare come la test automation viene adottata in ambienti diversi

2.2 Processo di Valutazione per Selezionare gli Strumenti e le Strategie Adatti

TAE-2.2.1 (K4) Analizzare un system under test per determinare la test automation solution appropriata

TAE-2.2.2 (K4) Illustrare i risultati tecnici della valutazione di uno strumento

2.1 Comprendere la Configurazione di un'Infrastruttura per Consentire la Test Automation

2.1.1 Descrivere le esigenze di configurazione di un'infrastruttura che consenta l'implementazione della test automation

La testabilità del SUT (cioè la disponibilità di interfacce software che supportano il testing, ad esempio per consentire 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. Questo lavoro è generalmente svolto da un architetto del software, poiché la testabilità è un requisito non funzionale del sistema, spesso con il coinvolgimento di un TAE per identificare le aree specifiche in cui è possibile apportare miglioramenti.

Per una migliore testabilità del SUT si possono utilizzare diverse soluzioni che hanno esigenze di configurazione diverse, ad esempio:

- Identificatori di accessibilità
 - I diversi framework di sviluppo possono generare questi identificatori automaticamente o gli sviluppatori possono impostarli manualmente
- Variabili di ambiente del sistema
 - Alcuni parametri dell'applicazione possono essere modificati per facilitare il testing attraverso l'amministrazione
- Variabili di deployment
 - Simili alle variabili di sistema, ma possono essere impostate prima di avviare il deployment

Progettare per la testabilità di un SUT consiste nei seguenti aspetti:

- Osservabilità: Il SUT deve fornire interfacce che diano informazioni sul SUT. I test case possono quindi utilizzare queste interfacce per determinare se i risultati effettivi corrispondono a quelli attesi
- Controllabilità: Il SUT deve fornire interfacce che possono essere utilizzate per eseguire azioni sul SUT. Può trattarsi di elementi dell'UI, di chiamate a funzioni, di elementi di comunicazione (ad esempio, i protocolli Transmission Control Protocol/Internet Protocol (TCP/IP) e Universal Serial Bus (USB)), o di segnali elettronici per interruttori fisici o logici sulle diverse variabili dell'ambiente
- Trasparenza dell'architettura: La documentazione di un'architettura deve fornire componenti e interfacce chiari e comprensibili che diano osservabilità e controllabilità a tutti i livelli di test e favoriscano la qualità.

2.1.2 Spiegare come la test automation viene adottata in ambienti diversi

I diversi tipi di test automatizzati possono essere eseguiti in ambienti diversi. Questi ambienti possono variare a seconda dei progetti e delle metodologie, e la maggior parte dei progetti ha uno o più ambienti da utilizzare per il testing. Da un punto di vista tecnico, questi ambienti possono essere creati usando container, software di virtualizzazione e altri approcci.

Una serie di possibili ambienti da considerare include i seguenti:

Ambiente di sviluppo locale

In un ambiente di sviluppo locale, il software viene inizialmente creato e i componenti vengono testati con l'automazione per verificarne l'idoneità funzionale. Nell'ambiente di sviluppo locale possono essere eseguiti molti tipi di test, tra cui testing di componente, testing della GUI e testing dell'application programming interface (API). È inoltre importante notare che, utilizzando un ambiente di sviluppo integrato (IDE) su un determinato computer, è possibile eseguire il testing white box per identificare tempestivamente problemi di codifica e di qualità.

Ambiente di build

Il suo scopo principale è produrre la build del software ed eseguire i test che ne verificano la correttezza in un ecosistema DevOps. Questo ambiente può essere un ambiente di sviluppo locale o un agente di continuous integration/continuous delivery (CI/CD) in cui i test di basso livello (cioè i test di componente e i test di integrazione dei componenti) e l'analisi statica possono essere eseguiti senza il deployment effettivo in altri ambienti.

Ambiente di integrazione

Dopo aver eseguito il testing di basso livello e l'analisi statica, la fase successiva utilizza un ambiente di integrazione dei sistemi. Qui una versione candidata al rilascio (release candidate) del SUT, completamente integrata con altri sistemi, è disponibile per essere testata. In questo ambiente è possibile eseguire una test suite completamente automatizzata, sia per i test dell'UI che per i test delle API. In questo ambiente non è rilevante il testing white-box, ma solo il testing black-box (con riferimento al testing di integrazione dei sistemi e/o di accettazione). È importante notare che questo è il primo ambiente in cui dovrebbe essere predisposto il monitoraggio per valutare cosa succede in background durante l'uso del SUT, in modo da consentire un'indagine efficiente di difetti/failure.

Ambiente di preproduzione

Un ambiente di preproduzione viene utilizzato soprattutto per valutare le caratteristiche di qualità non funzionali (ad esempio, l'efficienza delle prestazioni). Sebbene i test non funzionali possano essere eseguiti in qualsiasi ambiente, una particolare attenzione è rivolta in particolare alla preproduzione, perché si tratta dell'ambiente che dovrebbe essere più simile alla produzione. Spesso, in tale ambiente viene svolto il testing di accettazione dell'utente dagli stakeholder di business per verificare il prodotto finale, così come è possibile eseguire test suite automatizzate esistenti, se necessario. Anche questo ambiente viene monitorato.

Ambiente di produzione/operativo

Un ambiente di produzione può essere utilizzato per valutare le caratteristiche di qualità funzionali e non funzionali in tempo reale, mentre gli utenti interagiscono con il sistema distribuito, grazie al monitoraggio e ad alcune best practice che consentono di eseguire il testing in produzione (ad esempio, canary release, blue/green deployment e testing A/B).

2.2 Processo di Valutazione per Selezionare gli Strumenti e le Strategie Adatti

2.2.1 Analizzare un system under test per determinare la test automation solution appropriata

I SUT possono essere anche molto diversi tra loro, ma ci sono vari fattori e caratteristiche generali che possono essere analizzati per ottenere una test automation solution (TAS) di successo. Durante l'indagine di un SUT, i TAE devono raccogliere i requisiti tenendo conto del suo ambito e delle sue funzionalità. Diversi tipi di applicazioni (ad esempio, web service, mobile e web) necessitano di diversi tipi di test automation da un punto di vista tecnico. L'indagine può essere fatta - e lo si raccomanda - in collaborazione con altri stakeholder (ad esempio, tester manuali, stakeholder di business e business analyst) per identificare il maggior numero possibile di rischi e le relative mitigazioni, in modo da avere una test automation solution vantaggiosa per il futuro.

I requisiti per un approccio e un'architettura di test automation dovrebbero considerare quanto segue:

- Quali attività del processo di test dovrebbero essere automatizzate (ad esempio, test management, progettazione dei test, generazione dei test ed esecuzione dei test)
- Quali livelli di test dovrebbero essere supportati
- Quali tipi di test dovrebbero essere supportati
- Quali ruoli di testing e insieme di competenze dovrebbero essere supportati
- Quali prodotti software, linee e famiglie di prodotti dovrebbero essere supportati (ad esempio, per definire l'estensione e la durata della TAS implementata)
- Quali tipi di SUT devono essere compatibili con la TAS
- Disponibilità di dati di test e loro qualità
- Possibili metodi e modi per emulare i casi irraggiungibili (ad esempio, applicazioni di terze parti coinvolte)

2.2.2 Illustrare i risultati tecnici della valutazione di uno strumento

Dopo che il SUT è stato analizzato e i requisiti sono stati raccolti da tutti gli stakeholder, è probabile che esistano strumenti di test automation che soddisfino tali requisiti e che si possano prendere in considerazione. Potrebbe non esistere un unico strumento che soddisfi tutti i requisiti identificati e gli stakeholder dovrebbero riconoscere questa possibilità.

È utile raccogliere i risultati sui possibili strumenti e riflettere sui vari requisiti diretti e indiretti in una tabella di confronto. L'obiettivo della tabella di confronto è quello di consentire agli stakeholder di vedere le differenze tra gli strumenti in base ai requisiti specifici. La tabella di confronto elenca gli strumenti nelle colonne e i requisiti nelle righe. Le celle contengono informazioni sulle proprietà di ogni strumento rispetto a ogni requisito e sulle priorità.

In generale, gli strumenti di test automation dovrebbero essere valutati per determinare se soddisfano i requisiti identificati nel paragrafo precedente (2.2.1). I requisiti da considerare quando si valutano e si confrontano gli strumenti includono:

- Linguaggio/tecnologia dello strumento e strumenti dell'IDE
- La capacità di configurare uno strumento, se supporta diversi ambienti di test, configurazioni delle esecuzioni e se i corrispondenti valori possono essere impostati in modo dinamico o statico

- La capacità di gestire i dati di test all'interno dello strumento. La gestione dei dati di test potrebbe essere integrata con un repository centrale per il version control
- Per i diversi tipi di test, potrebbe essere necessario selezionare strumenti di test automation diversi
- La capacità di fornire funzionalità di reporting. Questo aspetto è importante per allinearsi ai requisiti di reporting dei test del progetto
- Capacità di integrarsi con altri strumenti utilizzati in un progetto o nell'organizzazione, come CI/CD, tracciamento delle attività, test management, reporting o altri strumenti
- Capacità di ampliare l'architettura di test complessiva e di valutare la scalabilità, la manutenibilità, la modificabilità, la compatibilità e l'affidabilità degli strumenti

Questa tabella di confronto è una buona fonte per determinare uno strumento o un insieme di strumenti da utilizzare per la test automation del SUT.

Il processo può variare a seconda di come si decide quale strumento/i utilizzare, ma la proposta dovrebbe essere presentata agli stakeholder appropriati per l'approvazione.

3 Test Automation Architecture - 210 minuti (K3)

Parole Chiave

behavior-driven development, cattura/riesecuzione, testing data-driven, generic test automation architecture, testing keyword-driven, scripting lineare, testing model-based, scripting strutturato, test adaptation layer, test automation framework, test automation solution, test harness, test script, testware, passo del test, test-driven development

Obiettivi di Apprendimento per il Capitolo 3:

3.1 Concetti di Progettazione Adottati nella Test Automation

TAE-3.1.1 (K2) Spiegare le caratteristiche principali di una test automation architecture

TAE-3.1.2 (K2) Spiegare come progettare una test automation solution

TAE-3.1.3 (K3) Applicare il layering nei test automation framework

TAE-3.1.4 (K3) Applicare i diversi approcci per automatizzare i test case

TAE-3.1.5 (K3) Applicare principi di progettazione e modelli di progettazione nella test automation

3.1 Concetti di Progettazione Adottati nella Test Automation

3.1.1 Spiegare le caratteristiche principali di una test automation architecture

Generic Test Automation Architecture (gTAA)

La gTAA è una concettualizzazione di una progettazione di alto livello che fornisce una visione astratta della comunicazione tra la test automation e i sistemi a cui la test automation è collegata, cioè il SUT, il project management, il test management e il configuration management (vedere la figura 1). Essa prevede inoltre le funzionalità necessarie da coprire durante la progettazione di una test automation architecture (TAA).

Le interfacce della gTAA descrivono quanto segue:

- L'interfaccia del SUT descrive la connessione tra il SUT e il TAF (vedere la sezione 3.1.3 relativa al test automation framework).
- L'interfaccia di project management descrive il progresso dello sviluppo di test automation.
- L'interfaccia di test management descrive la mappatura delle definizioni dei test case e dei test case automatizzati.
- L'interfaccia di configuration management descrive le pipeline di CI/CD, gli ambienti e il testware.

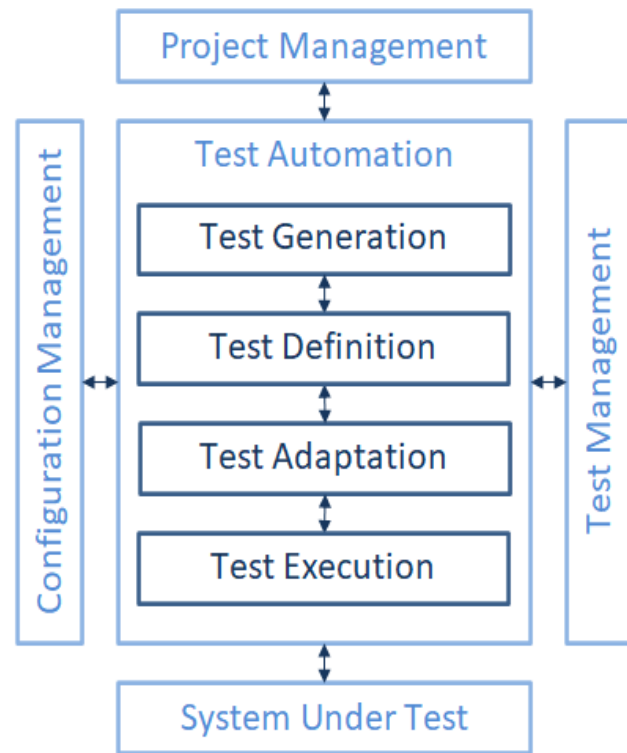


Figura 1: diagramma di gTAA

Le funzionalità fornite da strumenti e librerie di test automation

Le funzionalità fondamentali di test automation dovrebbero essere identificate e selezionate tra gli strumenti disponibili in base alle esigenze di un determinato progetto.

Test generation: Supporta la progettazione automatizzata di test case sulla base di un modello di test. Nel processo di generazione si possono utilizzare strumenti di testing model-based (vedere l'ISTQB CT-MBT Syllabus). La test generation è una funzionalità opzionale.

Test definition: Supporta la definizione e l'implementazione di test case e/o test suite, che opzionalmente possono essere derivati da un modello di test. Separa la definizione dei test dal SUT e/o dagli strumenti di test. Contiene gli strumenti per definire test di alto e basso livello, che vengono gestiti nei componenti dei dati di test, dei test case e delle librerie di test o in loro combinazioni.

Test execution: Supporta l'esecuzione e il logging dei test. Fornisce uno strumento di esecuzione dei test per eseguire automaticamente i test selezionati, un componente di logging e uno di reporting dei test.

Test adaptation: Fornisce le funzionalità necessarie per adattare i test automatizzati ai vari componenti o interfacce del SUT. Fornisce diversi adattatori per la connessione al SUT tramite API, protocolli e servizi.

3.1.2 Spiegare come progettare una test automation solution

Una test automation solution (TAS) è definita dalla comprensione dei requisiti funzionali, non funzionali e tecnici del SUT, degli strumenti esistenti o richiesti per implementare una soluzione. Una TAS è implementata con strumenti commerciali od open-source e può richiedere adattatori aggiuntivi specifici per il SUT.

La TAA definisce il progetto tecnico dell'intera TAS. Dovrebbe prevedere:

- Selezione degli strumenti di test automation e librerie specifiche per gli strumenti
- Sviluppo di plugin e/o componenti
- Identificazione i requisiti di connettività e interfaccia (ad esempio, firewall, database, uniform resource locator (URLs)/connessioni, mock/stub, code di messaggi e protocolli)
- Collegamento con gli strumenti di test management e di defect management
- Utilizzo di un sistema di version control e repository

3.1.3 Applicare il layering nei test automation framework

Framework di Test Automation

Il TAF è la base di una TAS. Spesso include una test harness, nota anche come test runner, e librerie di test, test script e test suite.

Layer del TAF

I layer del TAF definiscono un confine distinto di classi che hanno scopi simili, come i test case, il reporting dei test, il logging dei test, la crittografia e le test harness. Introducendo un layer per ogni singolo scopo, la progettazione può diventare complicata. Pertanto, si raccomanda di mantenere basso il numero di layer del TAF.

Test script

Il suo scopo è fornire un repository dei test case del SUT e delle annotazioni della test suite. Richiama i servizi del layer della business logic, che possono comprendere passi del test, flussi utente o chiamate ad API. Tuttavia, i test script non dovrebbero effettuare chiamate dirette alle core library.

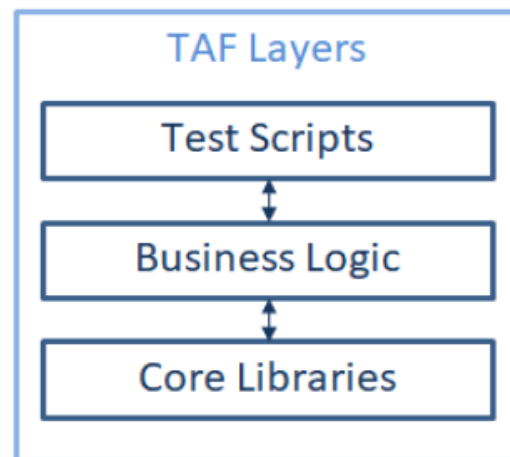


Figura 2: Test Automation Framework Layers

Business logic

Tutte le librerie dipendenti dal SUT sono memorizzate in questo layer. Queste librerie ereditano i file di classe delle core library o utilizzeranno le facade fornite da queste ultime (vedere la sezione 3.1.5 relativa all'ereditarietà e alle facade). Il layer della business logic viene utilizzato per impostare il TAF da eseguire sul SUT e le configurazioni aggiuntive.

Core libraries

Tutte le librerie indipendenti da qualsiasi SUT sono memorizzate in questo layer. Queste core library possono essere riutilizzate in qualsiasi tipo di progetto che condivida lo stesso stack di sviluppo.

Scalare la test automation

L'esempio seguente (Figura 3) mostra come le core library forniscono una base riutilizzabile per più TAF. Nel Progetto #1 ci sono due TAF costruiti sulle core library, mentre un progetto separato include le core library già esistenti per costruire il proprio TAF per testare l'App #3. Un TAE costruisce i TAF per il Progetto #1, mentre un secondo TAE costruisce il TAF per il Progetto #2.

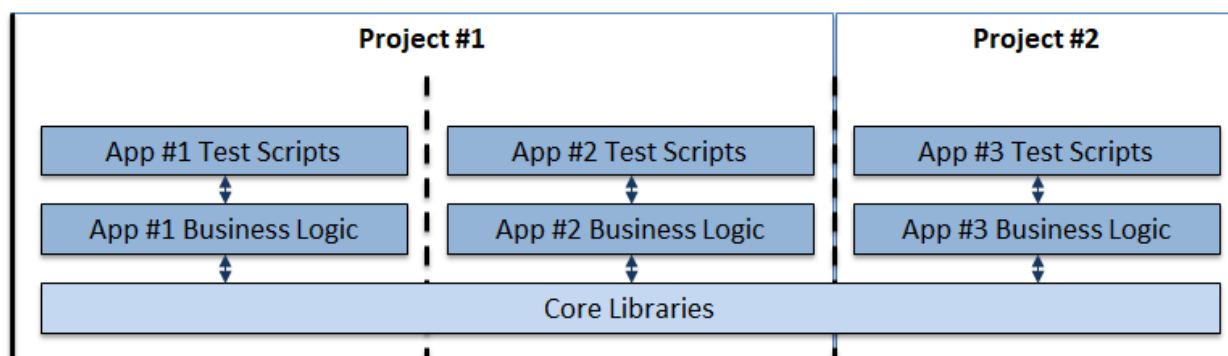


Figura 3: Applicazione delle core library a più TAF

3.1.4 Applicare i diversi approcci per automatizzare i test case

Esistono molti approcci di sviluppo che i team possono scegliere per produrre test case automatizzati. Questi possono includere linguaggi di scripting interattivi o linguaggi di programmazione compilati. I diversi approcci offrono vantaggi diversi in termini di automation e possono essere utilizzati in circostanze diverse. Sebbene test-driven development (TDD) e behavior-driven development (BDD) siano metodologie di sviluppo, se adottate correttamente, portano allo sviluppo di test case automatizzati.

Cattura/riesecuzione

La cattura/riesecuzione è un approccio che cattura le interazioni con il SUT durante l'esecuzione manuale di azioni su di esso. Questi strumenti producono test script durante la cattura e, a seconda dello strumento utilizzato, il codice di test automation può essere modificabile. Gli strumenti che non espongono il codice sono talvolta definiti di test automation no-code, mentre gli strumenti che espongono il codice sono definiti di test automation low-code.

Pro

- Inizialmente facile da configurare e utilizzare

Contro

- Difficile da mantenere, scalare ed evolvere
- Il SUT deve essere disponibile durante la cattura di un test case
- È fattibile solo per un ambito ridotto e un SUT che cambia raramente
- L'esecuzione del SUT catturato dipende fortemente dalla versione del SUT da cui è stata presa la cattura
- La registrazione di ogni singolo test case, che non prevede il riuso dei building block esistenti, richiede molto tempo.

Scripting lineare

Lo scripting lineare è un'attività di programmazione che non richiede librerie di test personalizzate realizzate da un TAE ed è utilizzato per scrivere ed eseguire i test script. Un TAE può utilizzare qualsiasi test script registrato da uno strumento di cattura/riesecuzione, che può quindi essere modificato.

Pro

- Facile da configurare e da utilizzare per iniziare a scrivere i test script
- Rispetto alla cattura/riesecuzione, i test script possono essere modificati più facilmente

Contro

- Difficile da mantenere, scalare ed evolvere
- Il SUT deve essere disponibile durante la cattura di un test case
- È fattibile solo per un ambito ridotto e un SUT che cambia raramente
- Rispetto alla cattura/riesecuzione, sono necessarie alcune conoscenze di programmazione

Scripting strutturato

Vengono introdotte librerie di test con elementi, passi del test e/o percorsi utente riutilizzabili. In questo approccio sono necessarie conoscenze di programmazione per la creazione e la manutenzione dei test script.

Pro

- Facile da mantenere, scalare, trasferire, adattare ed evolvere
- La business logic può essere separata dai test script

Contro

- Sono necessarie conoscenze di programmazione
- L'investimento iniziale nello sviluppo del TAF e nella definizione del testware richiede molto tempo

Test-driven development

I test case vengono definiti come parte del processo di sviluppo prima che una nuova funzionalità del SUT venga implementata. L'approccio di TDD è testare, codificare e refactor, anche noto come red, green e refactor. Lo sviluppatore identifica e crea un test case che fallirà (red). Poi sviluppa la funzionalità che soddisfa il test case (green). Il codice viene quindi sottoposto a refactoring per ottimizzarlo e rispettare i principi del codice pulito (clean code). Il processo continua con il test successivo e il successivo incremento di funzionalità.

Pro

- Semplifica lo sviluppo di test case a livello di componente
- Migliora la qualità e la struttura del codice
- Migliora la testabilità

- Rende più facile raggiungere la copertura del codice desiderata
- Riduce la propagazione dei difetti a livelli di test più alti
- Migliora la comunicazione tra sviluppatori, rappresentanti di business e tester
- Le user story che non sono state verificate con il testing delle GUI e il testing delle API possono raggiungere rapidamente i criteri di uscita seguendo il TDD

Contro

- Inizialmente ci vuole più tempo per abituarsi al TDD
- Non seguire correttamente il TDD può generare una falsa fiducia nella qualità del codice

Testing data-driven

Il testing data-driven (DDT) si basa sull'approccio di scripting strutturato. I test script vengono forniti insieme a dati di test (ad esempio, file .csv, .xlsx e dump di database). Ciò consente di eseguire più volte gli stessi test script con dati di test diversi.

Pro

- Permette un'espansione rapida e semplice dei test case attraverso l'alimentazione di dati
- Il costo dell'aggiunta di nuovi test automatizzati può essere ridotto in modo significativo
- I test analyst possono specificare i test automatizzati popolando uno o più file di dati di test che alimentano i test. In questo modo i test analyst hanno una maggiore libertà di specificare i test automatizzati, con meno dipendenza dai technical test analyst

Contro

- Può essere necessaria un'adeguata gestione dei dati di test

Testing keyword-driven

I test case del testing keyword-driven (KDT) sono un elenco o una tabella di passi del test derivati da keyword (parole chiave) e dai dati di test su cui le keyword operano. Le keyword sono definite dal punto di vista dell'utente. Questa tecnica è spesso basata sul DDT.

Pro

- I test analyst e i business analyst possono essere coinvolti nella creazione di test case automatizzati seguendo l'approccio KDT
- Il KDT può essere utilizzato anche per il testing manuale, indipendentemente dalla test automation. (vedere lo Standard ISO/IEC/IEEE 29119-5 relativo al Testing Keyword-Driven)

Contro

- L'implementazione e la manutenzione delle keyword è un compito complesso che i TAE devono svolgere e che può diventare impegnativo quando l'ambito cresce
- Crea un effort enorme per i sistemi più piccoli

Behavior-driven development

Il BDD utilizza un formato nel linguaggio naturale (ad esempio, given, when e then) per formulare criteri di accettazione che possono essere utilizzati come test case automatizzati e memorizzati in file di funzionalità. Uno strumento di BDD può quindi comprendere il linguaggio ed eseguire i test case.

Pro

- Migliora la comunicazione tra sviluppatori, rappresentanti di business e tester

- Gli scenari BDD automatizzati fungono da test case e garantiscono la copertura delle specifiche
- Il BDD può essere adottato per produrre più tipi di test su diversi livelli della piramide di test

Contro

- Ulteriori test case, tipicamente condizioni di test negative e casi limite, devono essere comunque definiti dal team, tipicamente da un test analyst o da un TAE
- Molti team pensano che BDD rappresenti soltanto un modo per scrivere test case in un linguaggio naturale e non coinvolgono i rappresentanti di business e gli sviluppatori nell'intero approccio
- L'implementazione e la manutenzione dei passi del test in linguaggio naturale è un compito complesso per i TAE da gestire
- Passi del test troppo complessi tendono a rendere il debugging un'attività difficile e costosa

3.1.5 Applicare principi di progettazione e modelli di progettazione nella test automation

La test automation è un'attività di sviluppo del software. Pertanto, i principi di progettazione (design principles) e i modelli di progettazione (design patterns) sono altrettanto importanti per un TAE quanto per uno sviluppatore di software.

Principi della programmazione object-oriented

I principi di programmazione object-oriented sono quattro: incapsulamento, astrazione, ereditarietà e polimorfismo.

Principi SOLID

È l'acronimo di single responsibility, open-closed, Liskov substitution, interface substitution e dependency inversion. Questi principi migliorano la leggibilità, la manutenibilità e la scalabilità del codice.

Modelli di progettazione

Tra i numerosi modelli di progettazione, i più importanti per i TAE sono tre.

Il modello facade nasconde i dettagli dell'implementazione per esporre solo ciò di cui i tester necessitano nel creare i test case e il modello singleton è spesso usato per assicurarsi che ci sia un solo driver che comunica con il SUT.

Nel modello page object, viene creato un file di classe a cui si fa riferimento come modello della pagina. Ogni volta che la struttura del SUT cambia, il TAE dovrà effettuare gli aggiornamenti in un solo punto, il locator all'interno del modello della pagina, invece di aggiornare i locator in ogni test case.

Il modello flow model è un'espansione del modello page object. Introduce una facade aggiuntiva rispetto ai modelli page object, che memorizza tutte le azioni tramite le quali l'utente interagisce con gli oggetti della pagina. Introducendo una progettazione a doppia facade, il modello flow model fornisce una migliore astrazione e manutenibilità, in quanto passi del test possono essere riutilizzati in più test script.

4 Implementazione della Test Automation - 150 minuti (K4)

Parole Chiave

rischio, test fixture

Obiettivi di Apprendimento per il Capitolo 4:

4.1 Sviluppo della Test Automation

TAE-4.1.1 (K3) Applicare linee guida che supportano lo svolgimento efficace di attività pilota e di deployment della test automation

4.2 Rischi Associati allo Sviluppo della Test Automation

TAE-4.2.1 (K4) Analizzare i rischi di deployment e pianificare strategie di mitigazione per la test automation

4.3 Manutenibilità della Test Automation Solution

TAE-4.3.1 (K2) Spiegare quali fattori supportano e influenzano la manutenibilità di una test automation solution

4.1 Sviluppo della Test Automation

4.1.1 Applicare linee guida che supportano lo svolgimento efficace di attività pilota e di deployment della test automation

È importante definire l'ambito di validazione di un pilota di test automation. Lo svolgimento di un progetto pilota non richiede molto tempo, ma il risultato può avere un impatto significativo sulla direzione che prenderà il progetto.

Sulla base delle informazioni raccolte sul SUT e dei requisiti del progetto, si dovrebbero valutare i seguenti aspetti nel definire linee guida per ottimizzare gli effort di test automation:

- Linguaggio/i di programmazione che verrà/verranno utilizzato/i
- Strumenti commerciali off-the-shelf/open-source adeguati
- Livelli di test da coprire
- Test case selezionati
- Approccio allo sviluppo dei test case

Sulla base dei punti sopra elencati, i TAE possono definire un approccio iniziale da seguire. In base ai requisiti, si possono creare diversi prototipi iniziali per mostrare i pro e i contro dei vari approcci. Da qui i TAE possono decidere quale percorso seguire.

Definire le scadenze è una parte importante per rispettare le schedulazioni e garantire il successo del pilota. Una raccomandazione comune è quella di monitorare periodicamente l'avanzamento del progetto pilota per identificare eventuali rischi e mitigarli.

Durante il pilota, si raccomanda anche di cercare di integrare la soluzione e il codice già implementato nella CI/CD. In questo modo si possono evidenziare i problemi in anticipo, siano essi nel SUT, nella TAS, piuttosto che nell'integrazione complessiva dei diversi strumenti all'interno dell'organizzazione.

Man mano che il numero di test case cresce, i TAE possono pensare di modificare la configurazione iniziale della CI/CD per eseguire i test in modi e tempi diversi.

Inoltre, durante il progetto pilota, è necessario valutare altri aspetti non tecnici, quali:

- La conoscenza e l'esperienza dei membri del team
- La struttura del team
- Licenze e regole organizzative
- Il tipo di testing pianificato e i livelli di test da coprire durante l'automazione dei test case

Una volta completato il progetto pilota, il suo esito ed effort dovrebbe essere valutato dai TAE e dai test manager per valutare il successo o il fallimento e prendere una decisione appropriata.

4.2 Rischi Associati allo Sviluppo della Test Automation

4.2.1 Analizzare i rischi di deployment e pianificare strategie di mitigazione per la test automation

L'interfacciamento del TAF con il SUT deve essere considerato come parte della progettazione architetturale. Poi si possono scegliere gli strumenti per il packaging, il logging dei test e la test harness.

Durante l'implementazione del pilota, è necessario considerare l'espansione e la manutenzione del codice di test automation. Questi sono fattori cruciali della fase di valutazione del pilota e possono influenzare fortemente la decisione finale.

Dal pilota si possono individuare diversi rischi di deployment:

- Aperture del firewall
- Utilizzo delle risorse (ad esempio, CPU e RAM)

È necessario prepararsi a rischi di deployment, come problemi di firewall, di utilizzo delle risorse, di connessione di rete e affidabilità. Questi aspetti non sono strettamente legati alla test automation, ma i TAE devono assicurarsi che siano soddisfatte tutte le condizioni per fornire quality gate affidabili e utili nel loro processo di sviluppo.

L'uso di dispositivi reali per la test automation su dispositivi mobile ne è un esempio. I dispositivi mobile devono essere accesi, avere una batteria sufficiente per funzionare durante il test, essere collegati a una rete e avere accesso al SUT.

I rischi tecnici di deployment possono includere:

- Packaging
- Logging
- Strutturazione dei test
- Aggiornamento

Packaging

Il packaging va preso in considerazione in quanto il version control della test automation, è tanto importante quanto quello del SUT. Potrebbe essere necessario caricare il testware in un repository da condividere all'interno dell'organizzazione, sia on-premise che in cloud.

Logging

Il logging dei test fornisce la maggior parte delle informazioni sui risultati dei test. Esistono diversi livelli di logging dei test e tutti sono utili per la test automation per vari motivi:

- Fatal: questo livello viene utilizzato per registrare gli eventi di errore che possono portare all'interruzione (abort) dell'esecuzione dei test
- Error: questo livello viene utilizzato quando una condizione o un'interazione fallisce e quindi fallisce anche il test case
- Warn: Questo livello viene utilizzato quando si verifica una condizione/azione inaspettata, ma ciò non interrompe il flusso del test case
- Info: Questo livello viene utilizzato per mostrare le informazioni di base su un test case e su ciò che accade durante l'esecuzione dei test
- Debug: Questo livello viene utilizzato per memorizzare dettagli specifici dell'esecuzione che generalmente non sono necessari per i log di base, ma sono utili durante l'indagine di un failure dei test
- Trace: Questo livello è simile a Debug, ma mira a fornire ancora più informazioni

Strutturazione dei test

La parte più importante della TAS è costituita dalla test harness e dalle test fixture in essa incluse, elementi che devono essere disponibili per l'esecuzione dei test. Le test fixture offrono libertà nel controllo dell'ambiente di test e dei dati di test. È possibile definire precondizioni e post-condizioni per l'esecuzione dei test e i test case possono essere raggruppati in test suite in diversi modi. Questi aspetti sono importanti

da valutare anche durante un progetto pilota. Inoltre, le test fixture consentono di creare test automatizzati ripetibili e atomici.

Aggiornamento

Uno dei rischi tecnici più comuni è rappresentato dagli aggiornamenti automatici delle test harness (ad esempio, degli agenti) e dai cambi di versione sui dispositivi. Questi rischi possono essere mitigati disponendo di alimentatori adeguati, connessioni di rete adatte e opportuni piani di configurazione dei dispositivi.

4.3 Manutenibilità della Test Automation Solution

4.3.1 Spiegare quali fattori supportano e influenzano la manutenibilità di una test automation solution

La manutenibilità è fortemente influenzata dagli standard di programmazione e dalle aspettative reciproche dei TAE.

Una regola d'oro consiste nel cercare di seguire i principi del codice pulito di Robert C. Martin (Robert C. Martin, "Clean Code: A Handbook of Agile Software Craftsmanship", 2008).

In breve, i principi del codice pulito evidenziano i seguenti punti:

- Utilizzare una naming convention comune per le classi, i metodi e le variabili, scegliendo nomi significativi
- Utilizzare una struttura di progetto logica e condivisa
- Evitare il ricorso all'hardcoding
- Evitare un numero eccessivo di parametri di input per i metodi
- Evitare metodi lunghi e complessi
- Utilizzare il logging
- Utilizzare i modelli di progettazione laddove utili e necessari
- Concentrarsi sulla testabilità

Le naming convention sono molto utili per identificare l'obiettivo di una determinata variabile. Avere nomi di variabili comprensibili come "loginButton", "resetPasswordButton" aiuta i TAE a capire quale componente utilizzare.

L'hardcoding è il processo che consiste nell'incorporare valori fissati all'interno del software, ovvero senza offrire la possibilità di variarli senza modificare il software stesso. Ciò può essere evitato tramite il testing data-driven, facendo in modo che i dati di test provengano da una sorgente comune che può essere mantenuta più facilmente. L'hardcoding riduce i tempi di sviluppo, ma non è consigliabile utilizzarlo perché i dati possono cambiare frequentemente, rendendone la manutenzione dispendiosa in termini di tempo. Si consiglia anche l'uso di costanti per le variabili che non si prevede cambino frequentemente. In questo modo, è possibile ridurre le sorgenti e le posizioni da mantenere.

Anche l'utilizzo di modelli di progettazione è altamente raccomandato. I modelli di progettazione, come descritto in 3.1.5, consentono di implementare un codice di test automation ben strutturato e ragionevolmente manutenibile, a patto che siano applicati in modo appropriato.

Per garantire un'elevata qualità del codice di test automation è consigliabile utilizzare analizzatori statici. I formattatori di codice, come quelli comunemente usati negli IDE, consentiranno di migliorare la leggibilità del codice di test automation.

Oltre ai principi del codice pulito, si raccomanda di utilizzare una struttura e una strategia di branching concordata nell'ambito del version control. L'uso di branch diversi per le funzionalità, i rilasci e le correzioni dei difetti è utile per ottenere una comprensione condivisa del contenuto di tali branch.

5 Strategie di Implementazione e Deployment per la Test Automation - 90 minuti (K3)

Parole Chiave

contract testing

Obiettivi di Apprendimento per il Capitolo 5:

5.1 Integrazione con le Pipeline di CI/CD

TAE-5.1.1 (K3) Applicare la test automation a diversi livelli di test all'interno delle pipeline

TAE-5.1.2 (K2) Spiegare il configuration management del testware

TAE-5.1.3 (K2) Spiegare le dipendenze della test automation per un'infrastruttura API

5.1 Integrazione con le Pipeline di CI/CD

5.1.1 Applicare la test automation a diversi livelli di test all'interno delle pipeline

Uno dei principali vantaggi della test automation è che i test implementati possono essere eseguiti senza supervisione, il che li rende candidati ideali per essere eseguiti all'interno delle pipeline. Ciò può avvenire attraverso le pipeline di CI/CD o la pipeline utilizzata per eseguire regolarmente i test.

I livelli di test sono solitamente integrati come segue:

- I test di configurazione per TAF/TAS, durante la build, possono essere considerati come una sottospecie di test di componente. Questi test vengono eseguiti durante la build di un progetto di test automation (TAF/TAS) e controllano che tutti i percorsi dei file utilizzati nei test script siano corretti, che i file esistano davvero e si trovino nei percorsi specificati.
- I test di componente fanno parte della fase di build della pipeline, in quanto vengono eseguiti sui singoli componenti (ad esempio, classi di libreria e componenti web). Essi fungono da quality gate per la pipeline e sono quindi una parte cruciale di una pipeline di continuous integration.
- I test di integrazione dei componenti possono far parte della pipeline di continuous integration se sono test di componenti di basso livello o del SUT. In tali casi, questi test vengono eseguiti insieme ai test di componente.
- I test di sistema possono spesso essere integrati in una pipeline di continuous deployment, dove agiscono come ultimo quality gate del SUT consegnato.
- I test di integrazione dei sistemi tra i diversi componenti del sistema sono spesso parte di una pipeline di continuous delivery come quality gate. Questi test di integrazione dei sistemi assicurano che i componenti del sistema sviluppati separatamente funzionino correttamente quando integrati.

Molti sistemi di continuous integration moderni distinguono tra le fasi di build e di deployment delle pipeline di continuous delivery. In questi casi, i test di componente e i test di integrazione dei componenti fanno parte della prima fase di build. Quando questa prima fase viene completata con successo (cioè, sia build che test), viene effettuato il deployment dei componenti/SUT.

Nel caso dei test di integrazione dei sistemi, del testing di sistema e del testing di accettazione, esistono due approcci principali per integrarli in tali pipeline:

1. I test case vengono eseguiti come parte della fase di deployment dopo il deployment del componente. Questo può essere vantaggioso, poiché, in base ai risultati dei test, il deployment può fallire ed esserne effettuato il rollback. Tuttavia, in questo caso, se i test devono essere rieseguiti, è necessario effettuare un nuovo deployment.
2. I test case vengono eseguiti come una pipeline separata, attivata dal successo del deployment. Questo può essere vantaggioso se ci si aspetta che test suite e codice di test automation differenti vengano eseguiti per ogni deployment. In questo caso, i test non agiscono come un quality gate. Di conseguenza, sono necessarie altre azioni, di solito manuali, per effettuare il rollback di un deployment che non ha avuto successo. In questo caso, si possono usare alcuni semplici test script automatizzati, che svolgono controlli sul deployment, per garantire il successo del deployment del SUT, ma questi test script automatizzati non verificano l'idoneità funzionale (functional suitability) in modo ampio.

Le pipeline possono essere utilizzate anche per altri scopi di test automation come ad esempio:

- Esecuzione periodica di diverse test suite: Una regression test suite può essere eseguita ogni notte (cioè la regressione notturna), soprattutto per le test suite di lunga durata, in modo che il team abbia un quadro chiaro della qualità del SUT al mattino.
- Esecuzione di test non funzionali: Sia come parte di una pipeline di continuous deployment, sia separatamente, per monitorare periodicamente alcune caratteristiche di qualità non funzionali del sistema, come l'efficienza delle prestazioni (performance efficiency).

5.1.2 Spiegare il configuration management del testware

Il configuration management è parte integrante della test automation, poiché spesso l'automazione viene eseguita su più ambienti di test e versioni del SUT.

Il configuration management nella test automation include:

- Configurazione dell'ambiente di test
- Dati di test
- Test suite/test case

Configurazione dell'ambiente di test

Ogni ambiente di test utilizzato nella pipeline di sviluppo può avere configurazioni diverse, come vari URL o credenziali. La configurazione dell'ambiente di test di solito viene memorizzata con il testware. Tuttavia, nel caso in cui la test automation venga utilizzata su più progetti o più TAF per lo stesso progetto, la configurazione dell'ambiente di test può far parte della core library comune o di un repository condiviso.

Dati di test

I dati di test possono anche essere specifici per l'ambiente di test o per la release e l'insieme di funzionalità del SUT. Come per la configurazione dell'ambiente di test, i dati di test sono di solito memorizzati in TAF più piccoli, ma possono essere utilizzati anche sistemi di gestione dei dati di test.

Test suite/test case

Una pratica comune è quella di creare diverse test suite dei test case, in base al loro scopo, come ad esempio lo smoke testing o il regression testing. Queste test suite vengono spesso eseguite in livelli di test separati, sfruttando pipeline e ambienti di test diversi.

Ogni release del SUT determina un insieme di funzionalità che include test case e test suite che valutano la qualità della release in questione. Esistono diverse opzioni nel testware per gestire questo aspetto:

- È possibile definire una configurazione di attivazione/disattivazione (toggle) delle funzionalità per ogni release o ambiente di test. Esistono test case e test suite per testare ogni funzionalità. La funzionalità "toggle" può essere usata nel testware per identificare quali test suite eseguire su una determinata release/ambiente di test.
- Il testware può anche essere rilasciato con il SUT utilizzando la stessa versione della release. In questo modo, c'è una corrispondenza esatta tra la versione del SUT e il testware che può testarlo. Questo tipo di release di solito viene implementato con un sistema di configuration management che utilizza tag o branch.

5.1.3 Spiegare le dipendenze della test automation per un'infrastruttura API

Quando si esegue la test automation delle API, è fondamentale avere le seguenti informazioni sulle dipendenze per costruire una strategia adeguata:

- Connessioni delle API: Comprendere la logica di business che può essere testata automaticamente e la relazione tra le API
- Documentazione delle API: Serve come base per la test automation con tutte le informazioni rilevanti (ad esempio, parametri, header e tipi distinti di richiesta-risposta degli oggetti)

Il testing automatizzato integrato delle API può essere effettuato sia dagli sviluppatori che dai TAE. Tuttavia, con l'approccio shift left si raccomanda di supportare e distribuire il testing tra diversi livelli. Nell'ISTQB CTFL Syllabus sono menzionati il testing di integrazione dei componenti e il testing di integrazione dei sistemi, che possono essere estesi con una best practice chiamata contract testing.

Contract testing

Il contract testing è un tipo di testing di integrazione che verifica che i servizi possano comunicare tra loro e che i dati condivisi tra i servizi siano coerenti con una serie di regole specificate. L'uso del contract testing fornisce la compatibilità tra due sistemi separati (ad esempio, due microservizi) focalizzandosi sulla comunicazione tra essi. Va oltre la validazione dello schema (schema validation), richiedendo a entrambe le parti di raggiungere un consenso sull'insieme di interazioni consentite, prevedendone un'evoluzione nel tempo. Cattura le interazioni che vengono scambiate tra ogni servizio, memorizzandole in un contratto che può essere utilizzato per verificare che entrambe le parti ne rispettino i termini. Uno dei principali vantaggi di questo tipo di test è costituito dalla possibilità di individuare precocemente nel SDLC eventuali difetti derivanti dai servizi sottostanti, facilitando l'identificazione della loro origine.

Nell'approccio consumer-driven al contract testing, è il consumer a definire le proprie aspettative, determinando come il provider debba rispondere alle richieste provenienti dal consumer stesso. Nell'approccio provider-driven, invece, è il provider a creare il contratto nel quale descrive il funzionamento dei propri servizi.

6 Reporting e Metriche della Test Automation - 150 minuti (K4)

Parole Chiave

misurazione, metrica, test log, test progress report, passo del test

Obiettivi di Apprendimento per il Capitolo 6:

6.1 Raccolta, Analisi e Reporting dei Dati della Test Automation

TAE-6.1.1 (K3) Applicare i metodi di raccolta dei dati dalla test automation solution e dal system under test

TAE-6.1.2 (K4) Analizzare i dati dalla test automation solution e dal system under test per comprendere meglio i risultati dei test

TAE-6.1.3 (K2) Spiegare come un test progress report viene costruito e pubblicato

6.1 Raccolta, Analisi e Reporting dei Dati della Test Automation

6.1.1 Applicare i metodi di raccolta dei dati dalla test automation solution e dal system under test

I dati possono essere raccolti dalle seguenti fonti:

- Log del SUT
 - UI web/mobile
 - API
 - Applicazioni
 - Server web
 - Server di database
- Log del TAF log per fornire un audit trail
- Log delle build
- Log dei deployment
- Log di produzione per monitorare i dati in produzione (vedi l'ISTQB CT-PT Syllabus, sezione 2.3)
 - Monitoraggio delle prestazioni in produzione per eseguire l'analisi delle tendenze
 - Log dei performance test in un ambiente di test dedicato al performance testing (ad esempio, load, stress, e spike testing).
- Screenshot e registrazioni dello schermo (nativi dello strumento di automazione o di terze parti)

Poiché una TAS ha come nucleo un testware automatizzato, quest'ultimo può essere migliorato per registrare informazioni sul suo utilizzo. I miglioramenti del testware apportati al testware sottostante possono essere utilizzati da tutti i test script automatizzati di livello superiore. Ad esempio, il miglioramento del testware sottostante per registrare l'ora di inizio e di fine dell'esecuzione del test può essere applicato a tutti i test.

Funzionalità della test automation che supportano la misurazione e la generazione di test report

I linguaggi di scripting di molti strumenti di test supportano la misurazione e il reporting e attraverso funzionalità che possono essere utilizzate per registrare e documentare informazioni prima, durante e dopo l'esecuzione dei test di singoli test e di intere test suite.

Il reporting dei test relativo a ogni esecuzione deve disporre di una funzionalità di analisi che consideri i risultati dei test delle precedenti esecuzioni, in modo da evidenziare eventuali tendenze, come ad esempio le variazioni del tasso di successo dei test.

La test automation richiede in genere di automatizzare sia l'esecuzione dei test che la verifica dei test, quest'ultima tramite il confronto di elementi specifici dei risultati effettivi con i risultati attesi. Questo confronto viene effettuato al meglio da uno strumento di test che utilizza le asserzioni. È necessario considerare il livello di informazione che viene riportato come risultato di questo confronto. È importante che lo stato del test sia determinato correttamente (cioè, superato o fallito). Nel caso di uno stato fallito, saranno necessarie ulteriori informazioni sulla causa del fallimento (ad esempio, screenshot).

Le differenze tra i risultati effettivi e i risultati attesi di un test non sono sempre chiare e il supporto dello strumento può essere di grande aiuto nel definire confronti che ignorino le differenze attese, come date e orari, evidenziando invece le differenze inattese.

Logging dei test

I test log sono una fonte spesso utilizzata per analizzare potenziali difetti all'interno della TAS e del SUT. Nella sezione seguente sono riportati esempi di logging dei test, suddivisi per TAS e SUT.

Logging della TAS

Il contesto determina se il TAF o l'esecuzione dei test è responsabile del logging delle informazioni, che dovrebbero includere quanto segue:

- Quale test case è attualmente in fase di esecuzione, compresi gli orari di inizio e di fine.
- Lo stato dell'esecuzione dei test perché, mentre i failure possono essere facilmente identificati nei test log, il TAF dovrebbe avere anche questa informazione e dovrebbe riportarla attraverso una dashboard. Lo stato di esecuzione dei test può essere superato, fallito o un failure della TAS. Lo stato può talvolta essere inconcludente ed è importante per un'organizzazione fornire definizioni chiare e coerenti. Un failure della TAS si applica a situazioni in cui il difetto non è presente nel SUT.
- Dettagli di basso livello del test log (ad esempio, logging di passi del test significativi), comprese le informazioni sulla tempistica.
- Informazioni dinamiche sul SUT (ad esempio, perdite di memoria) che il test case è stato in grado di identificare con l'aiuto di strumenti di terze parti. I risultati effettivi e i failure dovrebbero essere registrati con la test suite che è stata eseguita quando è stato rilevato il failure.
- Nel caso di testing di affidabilità o di stress testing, in cui vengono eseguiti numerosi cicli di test, si dovrebbe registrare un contatore per determinare facilmente quante volte sono stati eseguiti i test case.
- Quando i test case hanno elementi casuali (ad esempio, parametri casuali o passi del test casuali nel testing delle transizioni di stato), il numero o le scelte casuali dovrebbero essere registrati.
- Tutte le azioni eseguite da un test case dovrebbero essere registrate in modo tale che i test log, o parti di essi, possano essere riprodotti per rieseguire il test con gli stessi passi e gli stessi tempi. Ciò è utile per riprodurre un failure identificato e per acquisire ulteriori informazioni. Le informazioni sull'azione del test case possono anche essere registrate dal SUT al fine di cercare di riprodurre i failure identificati dal cliente. Se un cliente esegue una test suite, le informazioni del test log vengono acquisite e possono essere riprodotte dal team di sviluppo durante la risoluzione di un difetto.
- Gli screenshot possono essere salvati durante l'esecuzione dei test per essere utilizzate successivamente durante la root cause analysis.
- Ogni volta che l'esecuzione di una test suite produce un failure, la TAS dovrebbe assicurarsi che tutte le informazioni necessarie per analizzare il difetto siano disponibili/memorizzate, così come qualsiasi informazione riguardante la continuazione dell'esecuzione, se applicabile. La TAS dovrebbe salvare tutti i crash dump e i stack trace associati. Inoltre, tutti i test log che possono essere sovrascritti (ad esempio, i buffer ciclici sono spesso utilizzati per i test log sul SUT) dovrebbero essere archiviati dove saranno disponibili per analisi successive.
- L'uso dei colori può aiutare a distinguere i diversi tipi di informazioni del test log (ad esempio, i difetti in rosso e le informazioni sullo stato di avanzamento in verde).

Logging del SUT

La correlazione tra i risultati della test automation e i log del SUT aiuta a identificare la root cause dei difetti nel SUT e nella TAS.

- Quando viene identificato un difetto nel SUT, tutte le informazioni necessarie per analizzarlo dovrebbero essere registrate, inclusi data e orario, posizione di origine del difetto e messaggi di errore.
- All'avvio di un sistema, le informazioni di configurazione dovrebbero essere registrate in un file, costituito, ad esempio, dalle diverse versioni del software/firmware, dalla configurazione del SUT e dalla configurazione del sistema operativo.
- Utilizzando la test automation, i log del SUT possono essere facilmente ricercabili. Un failure identificato nel test log dalla TAS dovrebbe essere facilmente identificabile nel test log del SUT e

viceversa, con o senza strumenti aggiuntivi. La sincronizzazione dei vari test log con un timestamp facilita la correlazione di ciò che si è verificato quando è stato segnalato un failure.

Integrazione con altri strumenti di terze parti (ad esempio, fogli di calcolo, XML, documenti, database e strumenti per report).

Quando le informazioni ottenute dall'esecuzione dei test case automatizzati vengono utilizzate in altri strumenti per la tracciabilità e il reporting (ad esempio, per aggiornare le informazioni sulla tracciabilità), è possibile fornire le informazioni in un formato adatto agli strumenti di terze parti. Ciò è spesso realizzato tramite funzionalità già presenti negli strumenti di test (ad esempio, formati di esportazione per il reporting dei test) o creando report personalizzati in un formato coerente con altri software.

Visualizzazione dei risultati dei test

I risultati dei test possono essere resi visibili mediante grafici. Si raccomanda l'uso di icone colorate, come i semafori, per indicare lo stato complessivo dell'esecuzione dei test/della test automation in modo che le decisioni possano essere prese in base alle informazioni riportate. Il management è particolarmente interessato ai riepiloghi visivi per visualizzare i risultati dei test, come supporto nel processo decisionale. Se sono necessarie ulteriori informazioni, possono comunque approfondire i dettagli.

6.1.2 Analizzare i dati dalla test automation solution e dal system under test per comprendere meglio i risultati dei test

Dopo l'esecuzione dei test è importante analizzare i risultati dei test, per identificare eventuali failure sia nel SUT che nella TAS. Per tale analisi, i dati raccolti dalla TAS rappresentano la fonte primaria e i dati raccolti dal SUT quella secondaria.

- Analizzare i dati dell'ambiente di test per supportare il corretto dimensionamento della test automation (ad esempio, nel cloud)
 - Cluster e risorse (ad esempio, CPU e RAM)
 - Esecuzione dei test con un singolo browser piuttosto che con più browser (cioè, cross-browser)
- Confrontare i risultati dei test delle precedenti esecuzioni dei test
- Determinare come utilizzare i web log per monitorare l'utilizzo del software

I failure nell'esecuzione dei test devono essere analizzati, poiché esistono potenziali problemi:

1. Verificare se lo stesso failure si è verificato nelle precedenti esecuzioni dei test. Potrebbe trattarsi di un difetto noto nel SUT o nella TAS. La TAS può essere costruita per registrare i risultati dei test storici dei test case, aiutando così ulteriormente l'analisi.
2. Se il difetto non è noto, identificare il test case e ciò che sta testando. Il test può essere autoesplicativo, oppure il test case può essere identificato nel sistema di test management, in base al suo ID registrato con l'esecuzione dei test.
3. Trovare in quale passo del test si è verificato il failure. La TAS lo registra.
4. Analizzare le informazioni del test log sullo stato del SUT e sulla corrispondenza con i risultati attesi utilizzando screenshot, API e log della rete o qualsiasi log che mostri lo stato del SUT.
5. Se lo stato del SUT non è quello previsto, registrare un difetto nel sistema di defect management. Assicurarsi di includere tutte le informazioni necessarie sul difetto e i log che ne giustificano la registrazione come possibile difetto.

Quando si verifica un failure, potrebbe accadere che il risultato effettivo e il risultato atteso del SUT coincidano. In questo caso, molto probabilmente la TAS contiene un difetto che deve essere corretto, oppure è presente una discrepanza non visibile.

Un'altra situazione che può verificarsi è quella in cui l'ambiente di test non è disponibile durante l'esecuzione dei test, o lo è solo parzialmente. In questo caso, tutti i test case possono fallire, o con lo stesso difetto oppure, se parti del sistema sono fuori uso, esibendo failure apparentemente reali. Per identificare la root cause di tali difetti, è possibile analizzare i log del SUT, che mostreranno se ci sono state interruzioni nel funzionamento dell'ambiente di test al momento dell'esecuzione dei test.

L'implementazione sul SUT di log di audit per le interazioni degli utenti (ad esempio, sessioni di UI o chiamate ad API), aiuterebbe ad analizzare i risultati dei test. Solitamente, a ogni interazione viene aggiunto un ID univoco, utilizzato anche per ciascuna chiamata e integrazione successiva nel sistema. In questo modo, conoscendo l'ID univoco di una richiesta o interazione, è possibile osservare e tracciare il comportamento del sistema.

Questo ID univoco è solitamente chiamato 'correlation ID' o 'trace ID'. Può essere registrato dalla TAS per aiutare ad analizzare i risultati dei test.

6.1.3 Spiegare come un test progress report viene costruito e pubblicato

I test log forniscono informazioni dettagliate sui passi del test, sulle azioni da intraprendere e sulle risposte attese di un test case e/o di una test suite. Tuttavia, i soli test log non possono fornire una buona panoramica complessiva dei risultati dei test. Per questo è necessario disporre di una funzionalità di reporting dei test. Dopo l'esecuzione di una test suite, è necessario creare e pubblicare un sintetico test progress report. A questo scopo si può utilizzare un generatore di report.

Contenuto di un test progress report

Il test progress report deve contenere i risultati dei test, le informazioni sul SUT e la documentazione dell'ambiente di test in cui i test sono stati eseguiti, in un formato appropriato per ognuno degli stakeholder.

È necessario sapere quali test sono falliti e le ragioni del fallimento. Per facilitare la risoluzione di un problema, è importante conoscere la cronologia dell'esecuzione dei test e chi lo ha segnalato (cioè, in genere, la persona che ha creato il defect report o che lo ha aggiornato per ultima). La persona responsabile deve indagare sulla causa del failure, creare il defect report corrispondente, seguire la correzione del difetto e verificare che la correzione sia stata implementata correttamente.

Il reporting dei test viene utilizzato anche per diagnosticare eventuali failure dei componenti del TAF.

Pubblicare i test report

Il test report dovrebbe essere pubblicato per tutti gli stakeholder rilevanti. Può essere caricato su un sito web, nel cloud oppure on-premise, inviato a una mailing list o caricato su un altro strumento, come uno strumento di test management. Questo aiuta a garantire che i report vengano sottoposti a review e analizzati, soprattutto se le persone si sono iscritte per riceverli via e-mail o attraverso messaggi di chat inviati da un chatbot.

Un'opzione è quella di identificare le parti problematiche del SUT e di conservare uno storico dei test report, in modo da raccogliere statistiche sui test case o sulle test suite con frequenti regressioni per l'analisi delle tendenze.

Gli stakeholder a cui inviare report includono:

- Stakeholder del management
 - Ruoli tipici: solution o enterprise architect, project/delivery manager, program manager, test manager o test director
- Stakeholder operativi
 - Ruoli tipici: product owner/manager, rappresentante di business o business analyst

- Stakeholder tecnici
 - Ruoli tipici: team leader, scrum master, amministratore web, sviluppatore/amministratore di database, test leader, TAE, tester o sviluppatore

I test report possono variare nel contenuto o nei dettagli a seconda dei destinatari. Mentre gli stakeholder tecnici possono essere più interessati ai dettagli di basso livello, il management si concentrerà sulle tendenze, come ad esempio il numero di test case aggiunti dall'ultima esecuzione dei test, i cambiamenti nel rapporto di superamento-fallimento (pass-fail ratio) e l'affidabilità di TAS e SUT. Gli stakeholder operativi di solito pongono maggiore enfasi sulle metriche relative all'uso del prodotto.

Creazione di dashboard

I moderni strumenti di reporting offrono diverse opzioni di reporting attraverso dashboard, grafici colorati, raccolte dettagliate di log e analisi automatizzata dei test log. Sul mercato sono disponibili numerosi strumenti tra cui scegliere.

Questi strumenti supportano l'aggregazione dei dati da fonti quali i test log di esecuzione delle pipeline, gli strumenti di project management e i repository di codice. La visualizzazione dei dati fornita da questi strumenti aiuta gli stakeholder a notare le tendenze e a prendere decisioni di conseguenza. Queste tendenze possono includere cluster di difetti, aumento/diminuzione della propagazione dei difetti a determinati ambienti di test, degrado delle prestazioni del SUT e affidabilità delle build.

Analisi di intelligenza artificiale/machine learning dei test log

Negli ultimi anni, alcuni strumenti di test automation includono o si basano su algoritmi di machine learning (ML). L'analisi automatizzata di grandi quantità di dati nei test log aiuta il TAE a ridurre il tempo dedicato alla ricerca di locator non funzionanti, ad analizzare le cause dei failure dei test (cioè, se si tratta di un difetto nel SUT o nella TAS) e raggruppare i difetti comuni per il reporting dei test (vedere l'ISTQB CT-AI Syllabus).

7 Verifica della Test Automation Solution - 135 minuti (K3)

Parole Chiave

analisi statica

Obiettivi di Apprendimento per il Capitolo 7:

7.1 Verifica dell'Infrastruttura di Test Automation

TAE-7.1.1 (K3) Pianificare la verifica dell'ambiente di test automation, compresa la configurazione degli strumenti di test

TAE-7.1.2 (K2) Spiegare il comportamento corretto di un determinato test script e/o test suite automatizzata

TAE-7.1.3 (K2) Identificare dove la test automation produce risultati inattesi

TAE-7.1.4 (K2) Spiegare in che modo l'analisi statica può aiutare a migliorare la qualità del codice della test automation

7.1 Verifica dell'Infrastruttura di Test Automation

7.1.1 Pianificare la verifica dell'ambiente di test automation, compresa la configurazione degli strumenti di test

È comunque necessario verificare se l'ambiente di test automation e tutti gli altri componenti della TAS funzionano come previsto. Queste verifiche vengono effettuate, ad esempio, prima di avviare la test automation. Per verificare i componenti dell'ambiente di test automation, si possono seguire alcuni passi. Ognuno di questi è spiegato con maggior dettaglio di seguito.

Installazione, setup, configurazione e personalizzazione dello strumento di test

La TAS è composta da molti componenti, ognuno dei quali deve garantire prestazioni affidabili e ripetibili. Il cuore di una TAS è costituito da componenti eseguibili, dalle corrispondenti librerie di funzioni e dai file di dati e di configurazione di supporto. Il processo di configurazione di una TAS può variare dall'uso di script di installazione automatizzati al posizionamento manuale dei file nelle cartelle corrispondenti. Gli strumenti di test, analogamente a sistemi operativi e altri software, ricevono regolarmente dei service pack o possono includere dei componenti aggiuntivi (add-in) opzionali od obbligatori per poter garantire la compatibilità con un determinato ambiente del SUT.

L'installazione automatizzata o la copia da un repository presenta dei vantaggi. Può garantire che i test su SUT diversi vengano eseguiti con la stessa versione della TAS e, laddove appropriato, anche con la stessa configurazione della TAS. Gli aggiornamenti della TAS possono essere effettuati attraverso il repository. Le modalità d'uso del repository e il processo di aggiornamento a una nuova versione della TAS dovrebbero essere analoghi a quelli degli strumenti di sviluppo standard.

Ripetibilità nel setup/teardown dell'ambiente di test

Una TAS viene generalmente implementata per una varietà di sistemi, di server e per supportare le pipeline di CI/CD. Per garantire che la TAS funzioni correttamente in ogni ambiente di test, è necessario avere un approccio sistematico per caricare (load) e dismettere (unload) la TAS da ogni ambiente di test. Questo obiettivo è raggiunto con successo quando la costruzione e la ricostruzione della TAS non presenta differenze percepibili nel suo funzionamento all'interno e tra i vari ambienti di test. Il configuration management dei componenti della TAS assicura che una determinata configurazione possa essere creata in modo affidabile. Una volta raggiunto questo obiettivo, la documentazione dei vari componenti che compongono la TAS consentirà di fornire la conoscenza necessaria per comprendere quali aspetti della TAS possono essere influenzati o richiedere modifiche quando cambia l'ambiente del SUT.

Connettività con sistemi/interfacce interne ed esterne

Una volta installata una TAS in un determinato ambiente del SUT, e prima di utilizzare il SUT, è necessario eseguire una serie di controlli o impostare opportune precondizioni al fine di garantire che la connettività ai sistemi interni, ai sistemi esterni e alle interfacce sia disponibile. Ad esempio, è una buona pratica accedere ai server, lanciare gli strumenti di test automation, verificare che gli strumenti di test automation possano accedere al SUT, ispezionare manualmente le impostazioni di configurazione e assicurarsi che i permessi siano impostati correttamente per il logging dei test e il reporting dei test tra i sistemi. Stabilire le precondizioni per la test automation è essenziale per garantire che la TAS sia stata installata e configurata correttamente.

Testing dei componenti del TAF

Proprio come in qualsiasi progetto di sviluppo software, i componenti del TAF devono essere testati e verificati singolarmente. Ciò può includere test funzionali e non funzionali (come efficienza delle prestazioni e utilizzo delle risorse). Ad esempio, i componenti che forniscono la verifica degli oggetti sui sistemi GUI devono essere testati per un'ampia gamma di classi di oggetti per stabilire che la verifica degli oggetti funzioni correttamente. Allo stesso modo, i test log e i test report dovrebbero produrre informazioni accurate sullo stato della test automation e sul comportamento del SUT. Esempi di test non funzionale possono includere la comprensione del degrado delle prestazioni del TAF e dell'utilizzo delle risorse di sistema che possono indicare difetti come i memory leak e la mancanza di interoperabilità dei componenti all'interno e/o all'esterno del TAF.

7.1.2 Spiegare il comportamento corretto di un determinato test script e/o test suite automatizzata

Le test suite automatizzate devono essere testate per completezza, coerenza e comportamento corretto. Si possono applicare diversi tipi di controlli di verifica per assicurarsi che la test suite automatizzata sia disponibile in qualsiasi momento o per determinare che sia adatta all'uso.

Per verificare la test suite automatizzata si possono seguire alcuni passi. Questi includono:

- Controllare la composizione della test suite
- Verificare i nuovi test che utilizzano delle nuove funzionalità del TAF
- Considerare la ripetibilità dei test
- Considerare l'intrusività degli strumenti di test automatizzati

Ognuno di questi è spiegato in maggior dettaglio di seguito.

Controllare la composizione della test suite

Verificare la completezza (ad esempio, che tutti i test case abbiano tutti risultati attesi e i dati di test siano presenti) e la versione corretta del TAF e del SUT.

Verificare i nuovi test che utilizzano delle nuove funzionalità del TAF

La prima volta che una nuova funzionalità del TAF viene utilizzata nei test case, dovrebbe essere verificata e monitorata attentamente per assicurarsi che essa funzioni correttamente.

Considerare la ripetibilità dei test

Quando si ripetono i test, i risultati dei test dovrebbero essere sempre gli stessi. I test case nella test suite che non producono un risultato di test affidabile (ad esempio, a causa di race condition) dovrebbero essere rimossi dalla test suite automatizzata attiva e analizzati separatamente per trovare la root cause. In caso contrario, si rischia di perdere tempo ripetendo inutilmente queste esecuzioni per analizzare i fallimenti.

Considerare l'intrusività degli strumenti di test automatizzati

La TAS sarà spesso strettamente accoppiata ('tight coupling') al SUT. Questa caratteristica è intenzionale e definita a livello di progettazione, in quanto consente una migliore compatibilità per quanto riguarda il livello di interazione. Tuttavia, questo stretto accoppiamento può anche portare a risultati indesiderati. Ad esempio, quando la TAS si trova all'interno dell'ambiente del SUT, la funzionalità del SUT potrebbe esibire un comportamento diverso rispetto a quello osservato tramite i test condotti manualmente, con un potenziale impatto anche sulle prestazioni.

Un alto livello di intrusione può mostrare failure durante il testing che non sono evidenti in produzione. Se ciò causasse failure nei test automatizzati, la fiducia nella TAS potrebbe diminuire drasticamente. Gli sviluppatori potrebbero richiedere che i failure identificati dalla test automation siano riprodotti manualmente, se possibile, per agevolarne l'analisi.

7.1.3 Identificare dove la test automation produce risultati inattesi

Quando un test script fallisce o passa inaspettatamente, è necessario eseguire la root cause analysis. Ciò comprende l'ispezione dei test log, dei dati sulle prestazioni, del setup e teardown del test script.

È inoltre utile eseguire alcuni test isolati. I failure intermittenti sono più difficili da analizzare. Il difetto può essere nel test case, nel SUT, nel TAF, nell'hardware o nella rete. Il monitoraggio delle risorse di sistema può fornire indizi sulla root cause. L'analisi dei file del test log del test case, del SUT e del TAF può aiutare a identificare la root cause del difetto. Può essere necessario anche il debugging. Per aiutare a identificare la root cause può essere necessario il supporto di un test analyst, di un business analyst, di uno sviluppatore o di un ingegnere di sistema.

È importante verificare che tutte le asserzioni siano presenti, poiché la mancanza di asserzioni potrebbe portare a risultati di test inconcludenti.

7.1.4 Spiegare in che modo l'analisi statica può aiutare a migliorare la qualità del codice della test automation

L'analisi statica del codice può aiutare a trovare vulnerabilità e difetti nel codice di un programma. Questo può includere il SUT o il TAF.

Le scansioni automatizzate possono ispezionare il codice per mitigare i rischi. Ciò fornisce una review del SUT alla ricerca di difetti e aiuta a garantire il rispetto e l'applicazione degli standard di qualità del codice. Questa può anche essere considerata una tecnica di rilevamento proattivo dei difetti e svolge un ruolo significativo nelle implementazioni DevSecOps (cioè DevOps con enfasi sulla Sicurezza). Queste scansioni avvengono nelle prime fasi del SDLC tramite pipeline, per fornire feedback immediati ai team di sviluppo.

I difetti rilevati vengono solitamente classificati in base alla gravità (critica, alta, media o bassa), consentendo ai team di sviluppo di prioritizzare quali difetti correggere. Alcuni strumenti di analisi statica offrono anche suggerimenti di correzione del codice associati ai difetti che rilevano. Presenteranno ai team di sviluppo una copia delle linee di codice incriminate e offriranno agli sviluppatori un possibile rimedio da implementare. Inoltre, questi strumenti aiutano i TAE a misurare la qualità del codice, a suggerire aree in cui aggiungere commenti, a migliorare la progettazione del codice per un utilizzo ottimizzato delle risorse (ad esempio, tramite l'uso di blocchi try/catch e strutture di ciclo più efficienti) e a rimuovere chiamate scadenti alle librerie.

Poiché gli strumenti di test automation utilizzano linguaggi di programmazione, c'è il rischio di introdurre nel SDLC un codice di test automation inadeguato. Per fare un semplice esempio, una pratica comune quando si automatizzano i test è quella di avere un nome utente e una password. È possibile che un TAE includa erroneamente la password in chiaro (plain text) all'interno di uno o più test script.

Gli strumenti di analisi statica possono anche essere utili per il codice di test automation. Possono essere usati per analizzare il codice di test automation alla ricerca di violazioni della sicurezza, come ad esempio la presenza di password in chiaro all'interno del codice. Gli strumenti di analisi statica supportano molti linguaggi di programmazione, compresi quelli utilizzati dal software di test automation. Pertanto, è indispensabile che il TAE estenda le best practice di scansione del codice anche al codice di test automation. Anche se il codice di test automation non viene necessariamente distribuito con il software complessivo,

esistono chiaramente potenziali vulnerabilità qualora la password venga scoperta in un test script di automazione (vedere l'ISTQB CT-SEC Syllabus).

8 Miglioramento Continuo - 210 minuti (K4)

Parole Chiave

validazione dello schema, istogramma dei test

Obiettivi di Apprendimento per il Capitolo 8:

8.1 Opportunità di Miglioramento Continuo per la Test Automation

TAE-8.1.1 (K3) Scoprire opportunità per migliorare i test case attraverso la raccolta e l'analisi dei dati

TAE-8.1.2 (K4) Analizzare gli aspetti tecnici di una test automation solution già distribuita e fornire raccomandazioni per il miglioramento

TAE-8.1.3 (K3) Ristrutturare il testware automatizzato per allinearli agli aggiornamenti del SUT

TAE-8.1.4 (K2) Riassumere le opportunità per l'utilizzo degli strumenti di test automation

8.1 Opportunità di Miglioramento Continuo per la Test Automation

8.1.1 Scoprire opportunità per migliorare i test case attraverso la raccolta e l'analisi dei dati

La raccolta e l'analisi dei dati possono essere migliorate considerando vari tipi di dati con gli approcci descritti di seguito.

Istogramma del test

Un report visivo dei dati di test rappresentato come istogramma dei test fornisce potenziali aree di miglioramento per quanto riguarda le tendenze dei dati dei test case. I TAE possono identificare le possibili aree di miglioramento, poiché molti strumenti di CI/CD e di reporting dei test hanno la capacità di mostrare diversi risultati dei test e i rispettivi dati di test (ad esempio, log delle eccezioni, messaggi di errore e screenshot). L'istogramma dei test consente inoltre ai TAE di identificare e selezionare i test case fragili ed effettuare il refactoring con ulteriori miglioramenti o ripensando l'effettiva implementazione.

Intelligenza artificiale

Un'altra recente opportunità è l'utilizzo dell'Intelligenza Artificiale (AI) per supportare il testing e la test automation. Ad esempio, nei test case delle UI, i dati includono anche i valori dei locator delle UI che possono essere trattati come input. Recenti strumenti all'avanguardia mostrano la capacità di rilevare se un determinato locator è cambiato rispetto a quello utilizzato. Basandosi sul ML e sul riconoscimento delle immagini, possono identificare i nuovi selettori e utilizzare un algoritmo di auto-correzione (self-healing) per sistemare il test case e includere i locator modificati nel test report. Questo può accelerare fasi successive come il version control e la manutenzione del codice.

Validazione dello schema

La validazione dello schema (schema validation) può essere applicata nell'analisi dei dati delle API (ad esempio, le proprietà derivate dagli endpoint di destinazione) e nell'analisi dei database (ad esempio, le regole di validazione per i campi software, come i tipi di dati e gli intervalli di valori consentiti).

Con la validazione dello schema, la TAS è in grado di verificare se una risposta corrisponde alla specifica di business effettiva. Questo tipo di controllo può essere utilizzato per determinare se gli elementi di risposta obbligatori sono presenti nella risposta del servizio e se il loro tipo di oggetto corrisponde a quello definito nello schema. Nel caso di schema non compatibile, la soluzione restituisce la validazione effettiva che aiuta i TAE a identificare la root cause del problema.

Esempio: un'API ha sei elementi di risposta obbligatori che nella risposta devono essere stringhe. Con gli strumenti di validazione dello schema, non è necessario scrivere singole asserzioni per verificare se questi tipi siano stringhe e se i loro valori non siano nulli. La validazione dello schema si occuperà di questi controlli, rendendo il codice di test automation implementato molto più breve e aumentando l'efficienza nel rilevare i difetti nel servizio di backend.

8.1.2 Analizzare gli aspetti tecnici di una test automation solution distribuita e fornire raccomandazioni per il miglioramento

Oltre alle attività di manutenzione continuativa necessarie per mantenere la TAS sincronizzata con il SUT, ci sono molte opportunità per migliorare la TAS. Questi miglioramenti possono essere apportati per ottenere una serie di benefici, tra cui una maggiore efficienza (ad esempio, riducendo ulteriormente l'intervento manuale), una migliore facilità d'uso, funzionalità aggiuntive e un migliore supporto per il testing. La

decisione su come migliorare la TAS è influenzata dalle funzionalità che apportano il maggior valore a un progetto.

Le aree specifiche di una TAS che possono essere prese in considerazione per i miglioramenti includono scripting, esecuzione dei test, verifica, TAA, TAF, setup e teardown, documentazione, funzionalità della TAS, nonché aggiornamenti e upgrade della TAS. Queste aree sono descritte in maggior dettaglio qui di seguito.

Scripting

Le tecniche di scripting variano dallo scripting lineare all'approccio di testing data-driven, fino al più sofisticato approccio di testing keyword-driven, come descritto nella sezione 3.1.4. Potrebbe essere opportuno effettuare un upgrade dell'attuale tecnica di scripting della TAS per tutti i nuovi test automatizzati. La tecnica può essere adottata a tutti i test automatizzati esistenti o almeno a quelli che comportano il maggiore effort di manutenzione.

Un'altra area di miglioramento della TAS per i test script può focalizzarsi sulla loro implementazione. Ad esempio:

- Valutare la sovrapposizione tra test script/test case/passi del test per consolidare i test automatizzati. I test case che contengono sequenze di azioni simili non dovrebbero implementare questi passi del test più volte. Questi passi del test dovrebbero essere implementati da una funzione e aggiunti a una libreria, in modo da poter essere riutilizzati. Queste funzioni di libreria possono essere poi utilizzate da diversi test case. Ciò aumenta la manutenibilità del testware. Quando i passi del test non sono identici ma simili, può essere necessaria la parametrizzazione. Nota: questo è un approccio tipico del testing keyword-driven.
- Stabilire un processo di recupero dei failure per la TAS e per il SUT. Quando si verifica un failure durante l'esecuzione di una test suite, la TAS dovrebbe essere in grado di riprendersi da questa condizione per continuare con il prossimo test possibile. Quando si verifica un failure nel SUT, la TAS deve eseguire le azioni di ripristino necessarie sul SUT (ad esempio, un riavvio del SUT), ove possibile e pratico.
- Valutare i meccanismi di attesa per assicurarsi che venga utilizzato il tipo migliore. Esistono tre meccanismi di attesa comuni:
 - Le attese hardcoded (ad esempio, attendere un numero fissato di millisecondi), che possono essere la root cause di molti difetti di test automation, data l'imprevedibilità dei tempi di risposta del software.
 - L'attesa dinamica tramite polling (ad esempio, controllare periodicamente se un determinato cambiamento di stato o un'azione siano avvenuti) è molto più flessibile ed efficiente:
 - La TAS attende solo il tempo necessario e non viene sprecato tempo per i test
 - Se il processo richiede più tempo del previsto, il polling continuerà ad attendere fino a quando la condizione non sarà soddisfatta. È inoltre fondamentale includere un meccanismo di timeout, altrimenti il test potrebbe rimanere in attesa indefinitamente in caso di difetti.

Un'opzione ancora migliore è basata sulla sottoscrizione al meccanismo di eventi del SUT. Questa è molto più affidabile delle altre due opzioni, ma richiede che il linguaggio di scripting supporti la sottoscrizione ad eventi e che il SUT offra questi eventi alla TAS.

Anche in questo caso è necessario includere un meccanismo di timeout, altrimenti il test potrebbe rimanere in attesa indefinitamente in caso di difetti.

Esecuzione dei test

Quando una regression test suite automatizzata non viene completata perché l'esecuzione dei test richiede troppo tempo, può essere necessario eseguire i test in contemporanea su più ambienti di test, se possibile. Quando si utilizzano sistemi costosi per il testing, ci può essere un vincolo in base al quale il testing debba essere eseguito su un unico sistema di destinazione. Potrebbe essere necessario dividere la regression test suite in più parti, ed eseguire ognuna di esse in un periodo di tempo definito (ad esempio, in una sola notte). Un'ulteriore analisi della copertura della test automation potrebbe rivelare la presenza di duplicazioni. Rimuovere la duplicazione può ridurre i tempi di esecuzione dei test, consentendo di ottenere ulteriori efficienze. Nel caso della CI/CD, una buona pratica è quella di eseguire job batch in parallelo per ottimizzare il tempo di esecuzione dei test. Inoltre, è buona pratica schedulare job batch automatizzati per eseguire le diverse pipeline a una determinata ora, ad esempio ogni mattina, per ridurre le interazioni manuali e accelerare il processo di sviluppo.

Verifica

Prima di creare nuove funzioni di verifica, si raccomanda di adottare un insieme di metodi di verifica standard da utilizzare per tutti i test automatizzati. In questo modo si eviterà di re-implementare le azioni di verifica in più test. Quando i metodi di verifica non sono identici ma simili, l'uso della parametrizzazione aiuterà a consentire l'uso di una stessa funzione per più tipi di oggetti.

TAA

Potrebbe essere necessario modificare la TAA per supportare i miglioramenti della testabilità del SUT. Queste modifiche possono essere apportate all'architettura del SUT e/o alla TAA della TAS. Questo può fornire un importante miglioramento della test automation, ma può richiedere modifiche e investimenti significativi nel SUT/TAS. Ad esempio, se il SUT viene modificato per fornire API per il testing, la TAS dovrebbe essere sottoposta a un refactoring di conseguenza. L'aggiunta di questi tipi di funzionalità nelle fasi avanzate del SDLC può essere piuttosto costosa; è molto meglio pensarci all'inizio della test automation e nelle prime fasi del SDLC del SUT.

TAF

Spesso esistono nuove versioni delle core library utilizzate in un TAF. Talvolta, gli aggiornamenti sono importanti e la versione più recente non può essere immediatamente inserita nella lista delle dipendenze del TAF, poiché ciò potrebbe interrompere i test di molte squadre. Pertanto, è meglio eseguire prima un pilota e un'analisi degli impatti. Dopodiché, si può creare un piano per l'adozione di tali versioni. O tutti i team adottano la nuova versione delle core library contemporaneamente, aggiornando la dipendenza nel file di build del layer di core library, oppure ogni team decide individualmente quando aggiornarla nel proprio business logic layer. Alla fine, quando tutti i team sono pronti ad accettare la nuova versione delle core library, le dipendenze possono essere aggiornate nel layer di core library layer (vedere la sezione 3.1.3).

Setup e teardown

Le azioni e le configurazioni che si ripetono prima o dopo ogni test script o test suite dovrebbero essere spostate nei metodi di setup o teardown. In questo modo, tutte le modifiche che hanno un impatto sul codice

possono essere aggiornate in un unico punto, con conseguente riduzione degli effort di manutenzione. Ad esempio, le chiamate ai servizi web possono essere utilizzate per soddisfare le precondizioni o le post-condizioni dei test della UI (ad esempio, registrazione dell'utente, pulizia dell'utente e impostazione del profilo).

Documentazione

Questo comprende tutte le forme di documentazione, dalla documentazione della test automation (ad esempio, cosa fa il codice di test automation e come dovrebbe essere utilizzato), alla documentazione della TAS per l'utente, fino ai test report e ai test log prodotti dalla TAS.

Funzionalità della TAS

Aggiungere funzionalità e funzioni della TAS, come il reporting dettagliato dei test, i test log e l'integrazione con altri sistemi. È importante aggiungere solo nuove funzionalità che saranno effettivamente utilizzate. L'aggiunta di funzionalità non utilizzate aumenta solo la complessità e riduce affidabilità e manutenibilità.

Aggiornamenti e upgrade del TAF

Con l'aggiornamento o l'upgrade a nuove versioni del TAF, potrebbero diventare disponibili nuove funzioni che possono essere utilizzate dai test case o potrebbero essere corretti dei failure. Il rischio è che aggiornare il TAF, sia con un upgrade degli strumenti di test esistenti sia introducendone di nuovi, possa avere un impatto negativo sui test case esistenti. Prima di lanciare l'ultima versione dello strumento di test, è necessario testare la nuova versione dello strumento di test eseguendo dei test campione. I test campione dovrebbero essere rappresentativi dei test automatizzati di diversi SUT, di diversi tipi di test e, se appropriato, di diversi ambienti di test.

8.1.3 Ristrutturare il testware automatizzato per allinearli agli aggiornamenti del SUT

A seguito di una determinata serie di modifiche a un SUT esistente, sarà necessario aggiornare la TAS, compresi il TAF e le librerie di componente. Qualsiasi modifica, per quanto banale, può avere un ampio impatto negativo sull'affidabilità e sulle prestazioni della TAS.

Identificare le modifiche ai componenti dell'ambiente di test

Valutate le modifiche e i miglioramenti da apportare. Richiedono modifiche al testware, alle librerie di funzioni personalizzate o al sistema operativo? Ognuno di questi elementi ha un impatto sulle prestazioni della TAS. L'obiettivo generale è di garantire che i test automatizzati continuino a funzionare in modo efficiente. Le modifiche dovrebbero essere apportate in modo incrementale, con la mentalità del 'minimum viable product', in modo che l'impatto sulla TAS possa essere misurato attraverso una esecuzione limitata di test script. Una volta constatata l'assenza di effetti collaterali, le modifiche possono essere implementate pienamente. Un'esecuzione completa dei test di regressione è l'ultimo passo per verificare che la modifica non abbia avuto effetti negativi sui test script automatizzati. Durante l'esecuzione di questi test script di regressione, possono essere riscontrati dei failure. L'identificazione della root cause di questi failure (ad esempio, attraverso il reporting dei test, i test log e l'analisi dei dati di test) fornirà un mezzo per garantire che non derivino dall'attività di miglioramento della test automation.

Aumentare l'efficienza e l'efficacia delle core library delle funzioni della TAS

Man mano che una TAS matura, si scoprono nuovi modi per eseguire i compiti in modo più efficiente. Queste nuove tecniche (ad esempio, l'ottimizzazione del codice nelle funzioni e l'uso di nuove librerie del

sistema operativo) devono essere incorporate nelle core library delle funzioni utilizzate dal progetto attuale e dai progetti futuri.

Destinare più funzioni che agiscono sullo stesso tipo di controllo per il consolidamento

Gran parte di ciò che avviene durante un'esecuzione automatizzata dei test è l'interrogazione dei controlli nella GUI. Questa interrogazione serve a fornire informazioni su un controllo (ad esempio, visibile/non visibile, abilitato/non abilitato, misura e dimensioni, e dati). Con queste informazioni, un test automatizzato può selezionare un elemento da un elenco a discesa, inserire dati in un campo e leggere un valore da un campo. Esistono diverse funzioni che possono agire sui controlli per ottenere queste informazioni. Alcune funzioni sono estremamente specializzate, mentre altre sono di natura più generale. Ad esempio, può esistere una funzione specifica che agisce solo sugli elenchi a discesa. In alternativa, può esistere una funzione che lavora con diverse funzioni specificando una funzione come uno dei suoi parametri. Pertanto, un TAE può utilizzare diverse funzioni che possono essere consolidate in un numero inferiore di funzioni, ottenendo gli stessi risultati e riducendo al minimo la manutenzione.

Refactoring della TAA per adattarla alle modifiche del SUT

Nel corso del ciclo di vita di una TAS, sarà necessario apportare modifiche per accogliere i cambiamenti del SUT. Man mano che il SUT si evolve e matura, anche la TAA sottostante dovrà evolversi per garantire la capacità di supportare il SUT. Quando si estendono le funzionalità, occorre prestare attenzione affinché non vengano implementate in modo integrato, ma vengano invece analizzate e modificate nella TAA. In questo modo si garantirà che, quando le nuove funzionalità del SUT richiederanno test script aggiuntivi, saranno disponibili componenti compatibili per accogliere questi nuovi test automatizzati.

Convenzioni di denominazione e standardizzazione

Man mano che vengono introdotte delle modifiche, le convenzioni di denominazione per il nuovo codice di test automation e per le librerie di funzioni devono essere coerenti con gli standard precedentemente definiti (vedere la sezione 4.3.1).

Valutazione dei test script esistenti per la revisione/eliminazione del SUT

Il processo di modifica e miglioramento comprende anche una valutazione dei test script esistenti, del loro utilizzo e del loro valore continuo. Ad esempio, se alcuni test sono complessi e richiedono molto tempo per essere eseguiti, la loro scomposizione in test più piccoli può essere più praticabile ed efficiente. Mirare all'eliminazione dei test che vengono eseguiti in maniera non frequente o non vengono eseguiti affatto ridurrà la complessità della TAS e renderà più chiaro ciò che deve essere mantenuto.

8.1.4 Riassumere le opportunità per l'utilizzo degli strumenti di test automation

Oltre al testing vero e proprio, la test automation può aiutare in attività di test non specifiche come:

Configurazione e controllo dell'ambiente

Alcuni test script (ad esempio, creazione dei dati di test) possono essere sfruttati in un metodo di configurazione per creare diversi dati di test in un nuovo ambiente di test. In una situazione in cui è necessario creare utenti con più profili in base a diversi input di dati, un team può utilizzare un test script automatizzato per chiamare un endpoint di un servizio web che registra questi utenti. I test script possono avere il controllo sulla configurazione dell'infrastruttura del test e possono essere utilizzati in una pulizia successiva al processo. Ciò consente di risparmiare tempo e di garantire la presenza degli utenti corretti in ogni nuovo ambiente di test. Ad esempio, i diversi test log e altro testware possono essere rimossi

automaticamente da un ambiente di test, rendendo più efficiente la manutenzione e l'uso dell'ambiente di test.

Invecchiamento (aging) dei dati

La test automation può essere utilizzata per manipolare i dati di test nell'ambiente di test. Ad esempio, nei database, i campi delle date possono essere verificati e controllati per mantenerli aggiornati dal punto di vista dell'anno.

Generazione di screenshot e video

La maggior parte dei moderni strumenti di test automation dell'UI hanno una capacità integrata di creare screenshot o video in determinate condizioni e di memorizzarli. Con questi strumenti di test, i team possono supportare il business nella creazione di screenshot e video sull'utilizzo effettivo per la documentazione del rilascio del software o per scopi di marketing.

9 Riferimenti

Standard

Gli standard per la test automation includono, ma non si limitano a:

L'Automatic Test Markup Language (ATML) dell'IEEE (Institute of Electrical and Electronics Engineers) è costituito da

- IEEE Std 1671.1: Test Description
- IEEE Std 1671.2: Instrument Description
- IEEE Std 1671.3: UUT Description
- IEEE Std 1671.4: Test Configuration Description
- IEEE Std 1671.5: Test Adaptor Description
- IEEE Std 1671.6: Test Station Description
- IEEE Std 1641: Signal and Test Definition

IEEE Std 1636.1: Test Results

ISO/IEC/IEEE 29119-5 (2016) Software and systems engineering - Software testing - Part 5: Keyword-Driven Testing

ISO/IEC 30130:2016 (E) Software engineering – Capabilities of software testing tools

The Testing and Test Control Notation (TTCN-3) di ETSI (European Telecommunication Standards Institute) e ITU (International Telecommunication Union) è costituito da

- 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 Parametrization
- ES 202 785: Extensions: Behaviour Types
- ES 202 786: Extensions: Support of interfaces with continuous signals
- ES 202 789: Extensions: Extended TRI

L'UML Testing Profile (UTP) di OMG (Object Management Group) specifica i concetti di specifica dei test per

- Test Architecture
- Test Data
- Test Behavior
- Test Logging
- Test Management

Documenti ISTQB®

Identificatore	Riferimento
ISTQB-AL-TTA	ISTQB Certified Tester, Advanced Level Syllabus, Technical Test Analyst, Version 4.0, June 2021, disponibile su [ISTQB-Web]
ISTQB-FL	ISTQB Certified Tester, Foundation Level Syllabus, Version 4.0, April 2023, disponibile su [ISTQB-Web]
ISTQB-MBT	ISTQB Certified Tester, Model-Based Testing Syllabus, Version 1.0, October 2015, disponibile su [ISTQB-Web]
ISTQB-PT	ISTQB Certified Tester, Performance Testing Syllabus, December 2018, disponibile su [ISTQB-Web]
ISTQB-SEC	ISTQB Certified Tester, Security Tester Syllabus, Version 1.0, March 2016, disponibile su [ISTQB-Web]
ISTQB-TAS	ISTQB Certified Tester, Test Automation Strategy Syllabus, February 2024, disponibile su [ISTQB-Web]
ISTQB-Glossary	ISTQB Glossary of Terms, disponibile online su [ISTQB-Web]

Libri

Paul Baker, Zhen Ru Dai, Jens Grabowski e Ina Schieferdecker, "Model-Driven Testing: Using the UML Testing Profile", edizione Springer 2008, ISBN-10: 3540725628, ISBN-13: 978-3540725626
Efriede Dustin, Thom Garrett, Bernie Gauf, "Implementing Automated Software Testing: how to save time and lower costs while raising quality", Addison-Wesley, 2009, ISBN 0-321-58051-6
Efriede Dustin, Jeff Rashka, John Paul, "Automated Software Testing: introduction, management, and performance", Addison-Wesley, 1999, ISBN-10: 0201432870, ISBN-13: 9780201432879
Mark Fewster, Dorothy Graham, "Experiences of Test Automation: Case Studies of Software Test Automation", Addison-Wesley, 2012
Mark Fewster, Dorothy Graham, "Software Test Automation: Effective use of test execution tools", ACM Press Books, 1999, ISBN-10: 0201331403, ISBN-13: 9780201331400
Robert C Martin, "Clean Code: A Handbook of Agile Software Craftsmanship", 2008, ISBN-10: 9780132350884
James D. McCaffrey, ".NET Test Automation Recipes: A Problem-Solution Approach", APRESS, 2006 ISBN-13:978-1-59059-663-3, ISBN-10:1-59059-663-3
Daniel J. Mosley, Bruce A. Posey, "Just Enough Software Test Automation", Prentice Hall, 2002, ISBN-10: 0130084689, ISBN-13: 9780130084682
Manikandan Sambamurthy, "Test Automation Engineering Handbook", January 2023, ISBN: 9781804615492
Colin Willcock, Thomas Deiß, Stephan Tobies e Stefan Keil, "An Introduction to TTCN-3" Wiley, 2a edizione 2011, ISBN-10: 0470663065, ISBN-13: 978-0470663066

Articoli

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

10 Appendice A - Obiettivi di Apprendimento/Livello Cognitivo di Conoscenza

I seguenti obiettivi di apprendimento sono definiti come applicabili a questo Syllabus. Ogni argomento del Syllabus sarà esaminato in base al relativo obiettivo di apprendimento.

Gli obiettivi di apprendimento iniziano con un verbo di azione corrispondente al livello cognitivo di conoscenza, come elencato di seguito.

Livello 2: Comprendere (K2)

Il candidato è in grado di selezionare le ragioni o le spiegazioni delle affermazioni relative all'argomento ed è in grado di riassumere, confrontare, classificare e fornire esempi per il concetto del testing.

Verbi di azione: Classificare, confrontare, differenziare, distinguere, spiegare, fornire esempi, interpretare, riassumere.

Esempi	Note
Classificare gli strumenti di test in base al loro scopo e alle attività di test che supportano.	
Confrontare i diversi livelli di test.	Può essere utilizzato per cercare somiglianze, differenze o entrambe le cose.
Differenziare il testing dal debugging.	Cerca le differenze tra i concetti.
Distinguere tra rischi di progetto e rischi di prodotto.	Consente di classificare separatamente due (o più) concetti.
Spiegare l'impatto del contesto sul processo di test.	
Fornire esempi del perché il testing è necessario.	
Ricavare la root cause dei difetti da un determinato profilo di failure.	
Riassumere le attività del processo di review del prodotto di lavoro.	

Livello 3: Applicare (K3)

Il candidato è in grado di eseguire una procedura quando si trova di fronte a un compito che gli è familiare, oppure è in grado di selezionare la procedura corretta e applicarla a un determinato contesto.

Verbi di azione: Applicare, implementare, preparare, utilizzare

Esempi	Note
Applicare l'analisi ai valori limite per ricavare i test case da determinati requisiti.	Dovrebbe riferirsi a una procedura / tecnica / processo ecc.
Implementare i metodi di raccolta delle metriche per supportare i requisiti tecnici e di management.	
Preparare test di installabilità per le applicazioni mobile.	
Utilizzare la tracciabilità per monitorare l'avanzamento del test, in modo da poter verificare la completezza e la consistenza con gli obiettivi del test, la strategia di test e il test plan.	Potrebbe essere utilizzato in un LO che richiede che il candidato sia in grado di utilizzare una tecnica o una procedura. Simile ad "applicare".

Livello 4: Analizzare (K4)

Il candidato è in grado di separare le informazioni relative a una procedura o a una tecnica nelle sue parti costituenti per una migliore comprensione ed è in grado di distinguere tra fatti e inferenze.

Un'applicazione tipica è analizzare un documento, un software o una situazione di progetto, e proporre azioni appropriate per risolvere un problema o completare un compito.

Verbi di azione: Analizzare, analizzare in modo critico, delineare, prioritizzare, selezionare

Esempi	Note
Analizzare una determinata situazione di progetto per determinare quali tecniche di test black-box o basate sull'esperienza dovrebbero essere applicate per raggiungere obiettivi specifici.	Esaminabile solo in combinazione con un obiettivo misurabile dell'analisi. Dovrebbe essere della forma "Analizzare xxxx per xxxx" (o simile).
Prioritizzare i test case in una determinata test suite per l'esecuzione, in base ai rischi di prodotto correlati.	
Selezionare i livelli di test e i tipi di test appropriati per verificare un determinato insieme di requisiti.	Necessario quando la selezione richiede un'analisi.

Riferimento

(Per i livelli cognitivi degli obiettivi di apprendimento)

Anderson, L. W. and Krathwohl, D. R. (eds) (2001) A Taxonomy for Learning, Teaching, and

Assessing: A Revision of Bloom's Taxonomy of Educational Objectives, Allyn & Bacon

11 Appendice B - Matrice di Tracciabilità dei Business Outcome rispetto agli Obiettivi di Apprendimento

Questo capitolo elenca la tracciabilità tra i Business Outcome e gli Obiettivi di Apprendimento del Test Automation Engineering Specialist.

Business Outcome Test Automation Engineering Specialist			B 0 1	B 0 2	B 0 3	B 0 4	B 0 5	B 0 6	B 0 7	B 0 8	B 0 9	B 1 0	B 1 1	B 1 2
Capitolo 1	Introduzione e obiettivi della test automation													
1.1	Scopo della test automation													
1.1.1	Spiegare i vantaggi e gli svantaggi della test automation	2	X											
1.2	La test automation nel ciclo di vita dello sviluppo software													
1.2.1	Spiegare come la test automation si applica a diversi modelli del ciclo di vita dello sviluppo software	2		X										
1.2.2	Selezionare gli strumenti di test automation adatti per un determinato system under test	2		X										
Capitolo 2	Preparazione per la test automation													
2.1	Comprendere la configurazione di un'infrastruttura per consentire la test automation													
2.1.1	Descrivere le esigenze di configurazione di un'infrastruttura che consenta l'implementazione della test automation	2			X									
2.1.2	Spiegare come la test automation viene adottata in ambienti diversi	2			X									
2.2	Processo di valutazione per selezionare gli strumenti e le strategie adatti													
2.2.1	Analizzare un system under test per determinare la test automation solution appropriata	4				X								
2.2.2	Illustrare i risultati tecnici della valutazione di uno strumento	4				X								

Business Outcome Test Automation Engineering Specialist			B 0 1	B 0 2	B 0 3	B 0 4	B 0 5	B 0 6	B 0 7	B 0 8	B 0 9	B 1 0	B 1 1	B 1 2
Capitolo 3	Test Automation Architecture													
3.1	Concetti di progettazione adottati nella test automation													
3.1.1	Spiegare le capacità principali in una test automation architecture	2					X							
3.1.2	Spiegare come progettare una test automation solution	2					X							
3.1.3	Applicare il layering nei test automation framework	3					X							
3.1.4	Applicare i diversi approcci per automatizzare i test case	3					X							
3.1.5	Applicare principi di progettazione e modelli di progettazione nella test automation	3					X							
Capitolo 4	Implementazione della test automation													
4.1	Sviluppo della test automation													
4.1.1	Applicare linee guida che supportano lo svolgimento efficace di attività pilota e di deployment della test automation	3						X						
4.2	Rischi associati allo sviluppo della test automation													
4.2.1	Analizzare i rischi di deployment e pianificare strategie di mitigazione per la test automation	4							X					
4.3	Manutenibilità della test automation solution													
4.3.1	Spiegare quali fattori supportano e influenzano la manutenibilità della test automation solution	2								X				
Capitolo 5	Strategie di implementazione e deployment per la test automation													
5.1	Integrazione con le pipeline di CI/CD													
5.1.1	Applicare la test automation a diversi livelli di test all'interno delle pipeline	3									X			
5.1.2	Spiegare il configuration management del testware	2									X			

Business Outcome Test Automation Engineering Specialist			B 0 1	B 0 2	B 0 3	B 0 4	B 0 5	B 0 6	B 0 7	T 0 8	B 0 9	B 1 0	B 1 1	B 1 2
5.1.3	Spiegare le dipendenze della test automation per un'infrastruttura API	2									X			
Capitolo 6	Reporting e metriche della test automation													
6.1	Raccolta, analisi e reporting dei dati della test automation													
6.1.1	Applicare i metodi di raccolta dei dati dalla test automation solution e dal system under test	3										X		
6.1.2	Analizzare i dati dalla test automation solution e dal system under test per comprendere meglio i risultati dei test	4										X		
6.1.3	Spiegare come un test progress report viene costruito e pubblicato	2										X		
Capitolo 7	Verifica della test automation solution													
7.1	Verifica dell'infrastruttura di test automation													
7.1.1	Pianificare la verifica dell'ambiente di test automation, compresa la configurazione degli strumenti di test	3											X	
7.1.2	Spiegare il comportamento corretto di un determinato test script e/o test suite automatizzata	2											X	
7.1.3	Identificare dove la test automation produce risultati inattesi	2											X	
7.1.4	Spiegare in che modo l'analisi statica può aiutare a migliorare la qualità del codice della test automation	2											X	
Capitolo 8	Miglioramento continuo													
8.1	Opportunità di miglioramento continuo per la test automation													
8.1.1	Scoprire opportunità per migliorare i test case attraverso la raccolta e l'analisi dei dati	3												X

Business Outcome Test Automation Engineering Specialist			B 0 1	B 0 2	B 0 3	B 0 4	B 0 5	B 0 6	B 0 7	B 0 8	B 0 9	B 1 0	B 1 1	B 1 2
8.1.2	Analizzare gli aspetti tecnici di una test automation solution già distribuita e fornire raccomandazioni per il miglioramento	4												X
8.1.3	Ristrutturare il testware automatizzato per allinearli agli aggiornamenti del SUT	3												X
8.1.4	Riassumere le opportunità per l'utilizzo degli strumenti di test automation	2												X

12 Appendice C – Release Note

L'ISTQB® Test Automation Engineering Syllabus 2024 è un importante aggiornamento e riscrittura della versione 2016. Per questo motivo, non sono presenti release note dettagliate per capitolo e per paragrafo. Il contenuto è stato significativamente migliorato per il ruolo di test automation engineering, mentre il contenuto strategico è stato spostato in un nuovo ISTQB® Test Automation Strategy Syllabus 2024.

13 Appendice D - Termini Specifici del Dominio

Termine	Definizione
DevSecOps	Una metodologia che combina sviluppo, sicurezza e operazioni e utilizza un alto grado di automazione.
modello flow model	Una visione di alto livello del dominio di lavoro, dei suoi componenti e delle interconnessioni tra di essi.
percorsi utente	Una serie di passi che mostrano, dal punto di vista dell'esperienza della persona, come un utente interagisce con un system under test.
modello page object	Un modello di progettazione nella test automation per migliorare la manutenzione dei test e ridurre la duplicazione del codice.
version control	Un processo per registrare e archiviare versioni specifiche del codice sorgente.

14 Indice

Tutti i termini sono definiti nel Glossario ISTQB® (<http://glossary.istqb.org/>).

analisi statica; 20; 45; 65	test automation; 8; 9; 12; 13; 14; 15; 16; 17; 18; 22; 23; 24; 25; 26; 28; 30; 32; 34; 35; 36; 37; 38; 40; 41; 42; 44; 45; 46; 47; 48; 50; 51; 52; 53; 54; 55; 56; 57; 63; 64; 65; 66; 67; 68
architettura generica di test automation; 23	test automation engineer; 8
behavior-driven development; 23	test fixture; 30
cattura/riesecuzione; 23; 27	test harness; 23; 25; 32
DevSecOps; 48; 59; 68	test log; 39; 41
framework di test automation; 23	test progress report; 39; 43; 65
istogramma del test; 50; 51	test script; 23
metrica; 39	testabilità; 12; 18; 19; 27; 33; 53
misurazione; 39; 40	test-driven development; 23; 26
modello flow model; 29; 68	testing data-driven; 23; 52
modello page object; 68	testing della GUI; 18; 20
passo del test; 23; 39; 42; 52	testing delle API; 18; 38
percorso utente; 68	testing di contratti; 12; 35
rischio; 30; 48; 54	testing keyword-driven; 23; 52
script lineare; 23; 52	testing model-based; 23
script strutturato; 23	testware; 23; 54
soluzione di test automation; 12; 23; 30; 63; 64; 65; 66	validazione dello schema; 38; 50; 51
system under test; 14; 18; 39; 63; 65	version control; 25; 32; 51; 68
test adaptation layer; 23	