Certified Tester Advanced Level Technical Test Analyst (CTAL-TTA) Syllabus

V4.0

International Software Testing Qualifications Board

Avviso sul Copyright

Storico delle Revisioni

Versione	Data	Osservazioni
2012	19.10.2012	Rilascio dell'Assemblea Generale per la versione 2012
2019 V.10	18.10.2019	Rilascio dell'Assemblea Generale per la versione 2019
V4.0	28.06.2021	Revisione globale basato sulla versione Inglese v4.0 (include le revisioni v4.0 Alpha, Beta e finale)

Sommario

Riconoscimenti

La versione 2019 di questo documento è stata prodotta da una task force di professionisti del Working Group Advanced Level di International Software Testing Qualifications Board: Graham Bath (vicepresidente), Rex Black, Judy McKay, Kenji Onoshi, Mike Smith (presidente), Erik van Veenendaal.

Le seguenti persone hanno partecipato alla review, ai commenti e alla votazione della versione 2019 di questo Syllabus:

Dani Almog	Andrew Archer	Rex Black
Armin Born	Sudeep Chatterjee	Tibor Csöndes
Wim Decoutere	Klaudia Dusser-Zieger	Melinda Eckrich-Brájer
Peter Foldhazi	David Frei	Karol Frühauf
Jan Giesen	Attila Gyuri	Matthias Hamburg
Tamás Horváth	N. Khimanand	Jan te Kock
Attila Kovács	Claire Lohr	Rik Marselis
Marton Matyas	Judy McKay	Dénes Medzihradszky
Petr Neugebauer	Ingvar Nordström	Pálma Polyák
Meile Posthuma	Stuart Reid	Lloyd Roden
Adam Roman	Jan Sabak	Péter Sótér
Benjamin Timmermans	Stephanie van Dijck	Paul Weymouth

Questo documento è stato prodotto da una task force di professionisti del Working Group Advanced Level di International Software Testing Qualifications Board: Armin Born, Adam Roman, Stuart Reid.

La versione v4.0 aggiornata di questo documento è stata prodotta da una task force di professionisti del Working Group Advanced Level di International Software Testing Qualifications Board Advanced Level Working Group: Armin Born, Adam Roman, Christian Graf, Stuart Reid.

Le seguenti persone hanno partecipato alla review, ai commenti e alla votazione della versione inglese aggiornata V4.0 di questo Syllabus:

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

La task force ringrazia il team di review e i Board Nazionali per tutti i suggerimenti e i contributi.

Il presente documento è stato ufficialmente rilasciato dall'Assemblea Generale di ISTQB® nel Giugno 2021.

0. Introduzione a questo Syllabus

0.1 Scopo di questo Syllabus

Questo Syllabus costituisce la base per l'International Software Testing Qualification a Livello Advanced per i Technical Test Analyst. ISTQB® fornisce questo Syllabus come segue:

1. Ai Board Nazionali, per tradurlo nella loro lingua locale e per accreditare i Training Provider. I Board Nazionali possono adattare il Syllabus alle loro esigenze linguistiche particolari e modificare i riferimenti per adattarlo alle pubblicazioni locali.
2. Agli Organismi di Certificazione, per derivare le domande d'esame nella loro lingua locale, adattate agli Obiettivi di Apprendimento del Syllabus stesso.
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 autonomamente).
5. Alla comunità internazionale di Systems Engineering e Software Engineering, per promuovere la professione del testing dei sistemi e del software, e come base per libri e articoli.

ISTQB® può consentire ad altre entità di utilizzare questo Syllabus per altre finalità, a condizione che richiedano e ottengano anticipatamente autorizzazione scritta da parte dello stesso ISTQB®.

0.2 Il Certified Tester Advanced Level nel Software Testing

La certificazione Advanced Level si compone di tre Syllabi separati relativi ai seguenti ruoli:

- Test Manager
- Test Analyst
- Technical Test Analyst

ISTQB® Technical Test Analyst Advanced Level Overview è un documento separato [ISTQB_AL_TTA_OVIEW] che include le seguenti informazioni:

- Business Outcome di questo Syllabus
- Matrice di tracciabilità tra i Business Outcome e gli Obiettivi di Apprendimento
- Riassunto di ogni Syllabus

0.3 Obiettivi di Apprendimento Esaminabili e Livelli di Conoscenza

Gli Obiettivi di Apprendimento supportano i Business Outcome e vengono utilizzati per creare gli esami per il conseguimento della certificazione Advanced Technical Test Analyst.

I livelli di conoscenza degli Obiettivi di Apprendimento specifici e relativi ai livelli K2, K3 e K4 sono specificati all'inizio di ogni capitolo e sono classificati come segue:

- K2: Comprendere
- K3: Applicare
- K4: Analizzare

0.4 Aspettative di Esperienza

Alcuni degli Obiettivi di Apprendimento per il Technical Test Analyst presuppongono un'esperienza di base nelle seguenti aree:

- Concetti generali di programmazione
- Concetti generali sulle architetture di sistema

0.5 L'Esame Advanced Level Technical Test Analyst

L'esame per la certificazione Advanced Level Technical Test Analyst 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. Standard, libri e altri Syllabi ISTQB® sono inclusi come riferimenti, ma il loro contenuto non è esaminabile, al di là di quanto sintetizzato in questo Syllabus relativamente al loro contenuto.

L'esame è formato da 45 domande a scelta multipla. Per superare l'esame, deve essere ottenuto almeno il 65% dei punti totali.

Gli esami possono essere sostenuti come parte di un corso di formazione accreditato o intrapresi in modo indipendente (ad es. presso un centro d'esame o presso un esame pubblico). Il completamento di un corso di formazione accreditato non è un prerequisito per l'esame.

0.6 Requisiti di Ammissione per l'Esame

E' necessario aver conseguito la certificazione Certified Tester Foundation Level prima di sostenere l'esame di certificazione Advanced Level Technical Test Analyst.

0.7 Accreditamento dei Corsi

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 ad avere un esame ISTQB® come parte del corso stesso.

0.8 Livello di Dettaglio del Syllabus

Il livello di dettaglio di questo Syllabus consente di svolgere corsi di formazione ed esami consistenti a livello internazionale. Per ottenere questo obiettivo, il Syllabus consiste di:

- Un'indicazione di obiettivi didattici generali che descrivono i propositi dell'Advanced Level Technical Test Analyst
- Un elenco di termini che gli studenti devono essere in grado di richiamare
- Gli Obiettivi di Apprendimento per ogni area di conoscenza, che descrivono il risultato dell'apprendimento cognitivo da raggiungere
- Una descrizione dei concetti chiave, inclusi i riferimenti a determinate fonti, come letteratura universalmente accettata o standard.

Il contenuto del Syllabus non è una descrizione dell'intera area di conoscenza; riflette il livello di dettaglio che deve essere coperto dai corsi di formazione Advanced Level. Si focalizza su materiali che possono essere applicati a tutti i progetti software, utilizzando qualsiasi ciclo di vita dello sviluppo software. Il Syllabus non contiene Obiettivi di Apprendimento specifici relativi a un particolare modello di sviluppo software, ma descrive come questi concetti si applicano nello sviluppo software Agile, in altri tipi di modelli di sviluppo software iterativi e incrementali, e nei modelli di sviluppo software sequenziali.

0.9 Come è Organizzato questo Syllabus

Esistono sei capitoli con contenuti esaminabili. Nell'intestazione principale di ogni capitolo è specificato il tempo minimo necessario da dedicare al capitolo; non vengono forniti i tempi per i paragrafi interni al capitolo. Per i corsi di formazione accreditati, il Syllabus richiede un minimo di 20 ore di istruzione, distribuite nei sei capitoli come segue:

- Capitolo 1: I Compiti del Technical Test Analyst nel Testing Basato sul Rischio (30 minuti)
- Capitolo 2: Tecniche di Test White-Box (300 minutes)
- Capitolo 3: Analisi Statica e Dinamica (180 minuti)
- Capitolo 4: Caratteristiche di Qualità per il Testing Tecnico (345 minuti)
- Capitolo 5: Review (165 minuti)
- Capitolo 6: Strumenti di Test e Test Automation (180 minuti)

1. I Compiti del Technical Test Analyst nel Testing Basato sul Rischio - 30 minuti

Parole Chiave
rischio di prodotto, rischio di progetto, valutazione del rischio, identificazione del rischio, mitigazione del rischio, testing basato sul rischio

Obiettivi di Apprendimento per I Compiti del Technical Test Analyst nel Testing Basato sul Rischio

1.2 Attività del Testing Basato sul Rischio

TTA-1.2.1 (K2) Riassumere i fattori di rischio generici che il Technical Test Analyst deve normalmente prendere in considerazione

TTA-1.2.2 (K2) Riassumere le attività del Technical Test Analyst all'interno delle attività di test in un approccio basato sul rischio

1.1 Introduzione

Il Test Manager ha la responsabilità generale di stabilire e gestire una strategia di test basata sul rischio. Il Test Manager di solito richiederà il coinvolgimento del Technical Test Analyst per garantire che l'approccio basato sul rischio sia implementato correttamente.

I Technical Test Analyst lavorano all'interno del framework del testing basato sul rischio stabilito dal Test Manager per il progetto. Contribuiscono con la loro conoscenza dei rischi di prodotto tecnici relativi al progetto, come i rischi relativi alla sicurezza, all'affidabilità e alle prestazioni di sistema. Dovrebbero anche contribuire all'identificazione e alla gestione dei rischi di progetto associati all'ambiente di test, come l'acquisizione e il set-up degli ambienti di test per il testing di sicurezza, di affidabilità e delle prestazioni.

1.2 Attività del Testing Basato sul Rischio

I Technical Test Analyst sono attivamente coinvolti nelle seguenti attività di test basato sul rischio:

- Identificazione del rischio
- Valutazione del rischio
- Mitigazione del rischio

Queste attività vengono eseguite in modo iterativo durante il progetto per gestire nuovi rischi e modifiche di priorità, e per valutare e comunicare regolarmente lo stato del rischio.

1.2.1 Identificazione del Rischio

Considerando il campione più ampio possibile di stakeholder, è molto probabile che il processo di identificazione del rischio rilevi il maggior numero possibile di rischi significativi. Poiché i Technical Test Analyst possiedono competenze tecniche uniche, sono particolarmente adatti per condurre interviste agli esperti, eseguire brainstorming con i colleghi, e analizzare le loro esperienze per determinare le aree con maggiori probabilità di rischio di prodotto. In particolare, i Technical Test Analyst lavorano a stretto contatto con altri stakeholder, come sviluppatori, architetti, operations engineer, product owner, uffici di supporto locali, esperti tecnici e tecnici del service desk, per determinare le aree di rischio tecnico che impattano il prodotto e il progetto. Il coinvolgimento di altri stakeholder assicura che tutti i punti di vista vengano presi in considerazione ed è generalmente facilitato dai Test Manager.

I rischi che potrebbero essere identificati dal Technical Test Analyst sono tipicamente basati sulle caratteristiche di qualità di prodotto [ISO 25010] elencate al Capitolo 4 di questo Syllabus.

1.2.2 Valutazione del Rischio

L'identificazione del rischio riguarda l'identificazione del maggior numero possibile di rischi pertinenti, mentre la valutazione del rischio è lo studio di questi rischi identificati per classificare ogni rischio e determinare la probabilità e l'impatto ad esso associati.

La probabilità di un rischio di prodotto è generalmente interpretata come la probabilità di accadimento del failure nel sistema sotto test. Il Technical Test Analyst contribuisce a comprendere la probabilità di ogni rischio di prodotto tecnico, mentre il Test Analyst contribuisce a comprendere il potenziale impatto di business del problema nel caso si verifichi.

I rischi di progetto che diventano problemi possono impattare sul successo complessivo del progetto. In genere, è necessario considerare i seguenti fattori di rischio di progetto:

- Conflitto tra stakeholder sui requisiti tecnici
- Problemi di comunicazione a causa della distribuzione geografica dell'organizzazione di sviluppo

- Strumenti e tecnologia (comprese le relative competenze)
- Tempi, risorse e pressioni del management
- Mancanza di quality assurance nelle fasi inziali
- Elevata frequenza di modifiche dei requisiti tecnici

I rischi di prodotto che diventano problemi possono causare un numero maggiore di difetti. In genere, è necessario considerare i seguenti fattori di rischio di prodotto:

- Complessità della tecnologia
- Complessità del codice
- Quantità di modifiche nel codice sorgete (inserimenti, cancellazioni, modifiche)
- Elevato numero di difetti rilevati, relativi alle caratteristiche di qualità tecniche (storico dei difetti)
- Problemi tecnici di interfaccia e di integrazione

In base alle informazioni di rischio disponibili, il Technical Test Analyst propone una probabilità di rischio iniziale in base alle linee guida stabilite dal Test Manager. Il valore iniziale può essere modificato dal Test Manager quando tutti i punti di vista degli stakeholder sono stati presi in considerazione. L'impatto del rischio viene normalmente determinato dal Test Analyst.

1.2.3 Mitigazione del Rischio

Durante il progetto, i Technical Test Analyst influenzano il modo in cui il testing risponde ai rischi identificati. Questo generalmente comporta quanto segue:

- Progettare i test case per questi rischi che indirizzano le aree ad alto rischio e aiutare a valutare il rischio residuo
- Ridurre il rischio eseguendo i test case progettati e mettendo in atto appropriate misure di mitigazione e di contingency, come indicato nel Test Plan
- Valutare i rischi sulla base di informazioni aggiuntive raccolte durante lo svolgimento del progetto, e utilizzare tali informazioni per implementare misure di mitigazione che permettano di ridurre la probabilità di questi rischi

Il Technical Test Analyst collaborerà spesso con specialisti in aree come la sicurezza e l'efficienza delle prestazioni, per definire le misure di mitigazione del rischio e gli elementi della strategia di test. Informazioni aggiuntive possono essere ottenute dai Syllabi Specialist ISTQB®, come il Syllabus Security Testing [CTSL_SEC_SYL] e il Syllabus Performance Testing [CTSL_PT_SYL].

2. Tecniche di Test White-Box - 300 minuti

Parole Chiave

anomalia, condizione atomica, flusso di controllo, safety integrity level, testing delle API, testing delle condizioni multiple, testing delle decisioni, testing delle decisioni/condizioni modificate, testing delle istruzioni, tecnica di test white-box

Obiettivi di Apprendimento per le Tecniche di Test White-Box

2.2 Testing delle Istruzioni

TTA-2.2.1 (K3) Progettare test case per un determinato oggetto di test applicando il testing delle istruzioni per raggiungere un livello di copertura definito

2.3 Testing delle Decisioni

TTA-2.3.1 (K3) Progettare test case per un determinato oggetto di test applicando il testing delle decisioni per raggiungere un livello di copertura definito

2.4 Testing delle Decisioni/Condizioni Modificate

TTA-2.4.1 (K3) Progettare test case per un determinato oggetto di test applicando il testing delle decisioni/condizioni modificate per raggiungere la copertura completa delle decisioni/condizioni modificate (MC/DC)

2.5 Testing delle Condizioni Multiple

TTA-2.5.1 (K3) Progettare test case per un determinato oggetto di test applicando la tecnica di test delle condizioni multiple per raggiungere un livello di copertura definito

2.6 Testing dei Cammini Base *(è stato eliminato dalla versione V2.0 di questo Syllabus)*

TTA-2.6.1 è stato eliminato dalla versione V2.0 di questo Syllabus.

2.7 Testing delle API

TTA-2.7.1 (K2) Comprendere l'applicabilità del testing delle API e il tipo di difetti che è possibile rilevare

2.8 Selezionare una Tecnica di Test White-box

TTA-2.8.1 (K4) Selezionare una tecnica di test white-box appropriata in base a una determinata situazione progettuale

2.1 Introduzione

Questo capitolo descrive le tecniche di test white-box. Queste tecniche si applicano al codice e ad altre strutture con un flusso di controllo (flow control), come un flow chart del processo di business.

Ogni tecnica specifica consente di derivare sistematicamente i test case e si focalizza su un aspetto particolare della struttura. I test case generati dalle tecniche soddisfano i criteri di copertura che sono stati definiti come un obiettivo, e sono misurati rispetto a questo obiettivo. Il raggiungimento di una copertura completa (cioè 100%) non significa che l'insieme completo dei test sia completo, ma piuttosto che la tecnica utilizzata non suggerisce più ulteriori test utili per la struttura in esame.

Gli input dei test vengono generati per assicurare che un test case eserciti una particolare parte del codice (ad es. un'istruzione o un esito decisionale). Può essere sfidante determinare gli input dei test che eserciteranno una particolare parte del codice, specialmente se la parte del codice da esercitare è la parte finale di un sotto-cammino di un lungo flusso di controllo, con diverse decisioni. I risultati attesi sono identificati in base a una sorgente esterna a questa struttura, come una specifica di progettazione, una specifica dei requisiti o un'altra base di test.

In questo Syllabus sono prese in considerazione le seguenti tecniche:

- Testing delle istruzioni
- Testing delle decisioni
- Testing delle decisioni/condizioni modificate
- Testing delle condizioni multiple
- Testing delle API

Il Syllabus Foundation [ISTQB_FL_SYL] introduce il testing delle istruzioni e il testing delle decisioni. Il testing delle istruzioni si focalizza su come esercitare le istruzioni eseguibili nel codice, mentre il testing delle decisioni esercita gli esiti decisionali.

Le tecniche di test delle decisioni/condizioni modificate e delle condizioni multiple elencate sopra sono basate su predicati decisionali che contengono condizioni multiple, e rilevano tipi di difetti similari. Per quanto complesso possa essere un predicato decisionale, il risultato sarà valutato come VERO o FALSO, e questo determinerà il cammino da considerare nel codice. Viene rilevato un difetto quando il cammino previsto non viene intrapreso perché un predicato decisionale non viene valutato come previsto.

Fare riferimento a [ISO 29119] per maggiori dettagli sulla specifica, e su esempi, di queste tecniche.

2.2 Testing delle Istruzioni

Il testing delle istruzioni esercita le istruzioni eseguibili nel codice. La copertura viene misurata come il numero di istruzioni eseguite dai test diviso il numero totale di istruzioni eseguibili nell'oggetto di test, normalmente espressa come percentuale.

Applicabilità

Raggiungere la copertura completa delle istruzioni dovrebbe essere considerato il requisito minimo per tutto il codice da testare, sebbene in pratica questo non è sempre possibile.

Limitazioni/Difficoltà

Raggiungere la copertura completa delle istruzioni dovrebbe essere considerato il requisito minimo per tutto il codice da testare, sebbene in pratica questo non è sempre possibile a causa di vincoli sul tempo disponibile e/o sull'effort. Anche alte percentuali di copertura delle istruzioni possono non rilevare alcuni

difetti nella logica del codice. In molti casi, non è possibile raggiungere il 100% di copertura delle istruzioni a causa di codice non raggiungibile. Sebbene il codice irraggiungibile non sia generalmente considerato una buona pratica di programmazione, può accadere, ad esempio, che un'istruzione "switch" abbia tutti i casi gestiti esplicitamente anche se deve avere un "default case".

2.3 Testing delle Decisioni

Il testing delle decisioni esercita i risultati decisionali nel codice. Per fare questo, i test case seguono i flussi di controllo da un punto decisionale (ad es. in un'istruzione IF, esiste un flusso di controllo per l'esito vero e uno per l'esito falso; in un'istruzione CASE, possono esistere diversi possibili risultati, in un'istruzione LOOP esiste un flusso di controllo per l'esito vero della condizione loop e uno per l'esito falso).

La copertura viene misurata come il numero di esiti decisionali esercitati dai test diviso il numero totale di esiti decisionali nell'oggetto del test, normalmente espressa come percentuale. Si osservi che un singolo test case può esercitare diversi esiti decisionali.

Rispetto alle tecniche delle decisioni/condizioni modificate e delle condizioni multiple descritte di seguito, il testing delle decisioni considera l'intera decisione nel suo insieme e valuta solo gli esiti VERO e FALSO, indipendentemente dalla complessità della sua struttura interna.

Il testing dei rami è spesso usato al posto del testing delle decisioni, perché la copertura di tutti i rami e la copertura di tutte le decisioni possono essere raggiunte con gli stessi test. Il testing dei rami esercita i rami nel codice, dove un ramo è normalmente considerato essere un segmento del grafo del flusso di controllo. Per programmi senza decisioni, la definizione di copertura delle decisioni specificata porta a un risultato di copertura 0/0, che è indefinito, indipendentemente dal numero di test eseguiti, mentre un singolo ramo dal punto iniziale al punto finale (assumendo un solo punto iniziale e un solo punto finale) porterà ad una copertura dei rami pari al 100%. Per indirizzare la differenza tra le due misure, ISO29119-4 richiede che venga eseguito almeno un test sul codice senza decisioni, per raggiungere una copertura delle decisioni al 100%, in modo da ottenere una copertura delle decisioni al 100% e una copertura dei rami al 100% equivalente a quella che si ottiene con la maggior parte dei programmi. Molti strumenti di test che forniscono misure di copertura, incluse quelle utilizzate per i sistemi safety, utilizzano un approccio similare.

Applicabilità

Questo livello di copertura dovrebbe essere preso in considerazione quando il codice da testare è importante o addirittura critico (si vedano le tabelle nel paragrafo 2.7.2 per sistemi safety). Questa tecnica può essere utilizzata per il codice e per qualsiasi modello che prende in considerazione i punti decisionali, come i modelli dei processi di business.

Limitazioni/Difficoltà

Il testing delle decisioni non considera i dettagli di come viene presa una decisione con condizioni multiple e può fallire nel rilevare i difetti causati dalle combinazioni degli esiti delle condizioni.

2.4 Testing delle Decisioni/Condizioni Modificate

Rispetto al testing delle decisioni, che considera l'intera decisione nel suo insieme e valuta gli esiti VERO e FALSO, il testing delle decisioni/condizioni modificate considera come viene strutturata una decisione quando include condizioni multiple (dove una decisione è composta da un'unica condizione atomica, è semplicemente testing delle decisioni).

Ogni predicato decisionale è costituito da una o più condizioni atomiche, ciascuna delle quali valuta un valore Booleano. Questi sono combinati logicamente per determinare il risultato della decisione. Questa tecnica verifica che ciascuna delle condizioni atomiche influenzi in modo indipendente e corretto il risultato della decisione complessiva.

Il livello di copertura delle Decisioni/Condizioni modificate viene definito come MC/DC. Quando esistono decisioni che contengono condizioni multiple, questa tecnica fornisce un livello di copertura più forte rispetto alla copertura delle istruzioni e delle decisioni. Supponendo che ci siano N condizioni atomiche uniche e mutualmente indipendenti, MC/DC per una decisione può essere generalmente raggiunta esercitando la decisione per N+1 volte. Il testing delle decisioni/condizioni modificate richiede coppie di test che mostrano che una modifica di un risultato di una singola condizione atomica può influenzare in modo indipendente il risultato di una decisione. Si osservi che un singolo test case può esercitare diverse combinazioni di condizioni e quindi non è sempre necessario eseguire N+1 test case separati per raggiungere MC/DC.

Applicabilità

Questa tecnica viene utilizzata nelle industrie aerospaziali e automotive, e in altri settori industriali per sistemi safety-critical. Viene utilizzato quando nel testing del software un failure può causare una catastrofe. Il testing delle decisioni/condizioni modificate può essere una ragionevole via di mezzo tra il testing delle decisioni e il testing delle condizioni multiple (a causa del gran numero di combinazioni da testare). E' più rigoroso del testing delle decisioni ma richiede un numero minore di condizioni di test da esercitare rispetto al testing delle condizioni multiple, quando esistono diverse condizioni atomiche nella decisione.

Limitazioni/Difficoltà

Il raggiungimento di MC/DC può risultare complicato quando esistono occorrenze multiple della stessa variabile in una decisione con condizioni multiple. Quando questo si verifica, le condizioni possono essere “accoppiate”. In base alla decisione, può non essere possibile variare il valore di una condizione in modo tale che da sola causi il cambiamento del esito decisionale. Un approccio per indirizzare questo problema è specificare che vengono testate solo le condizioni atomiche disaccoppiate utilizzando il testing delle decisioni/condizioni modificate. Un altro approccio consiste nell'analizzare ogni decisione in cui si verifica l'accoppiamento.

Alcuni compilatori e/o interpreti sono progettati in modo tale da utilizzare un cortocircuito (ovvero, esibire un comportamento di short-circuiting) per la valutazione di un’istruzione di decisione complessa nel codice. Questo significa che il codice in esecuzione può non valutare un'intera espressione nel caso in cui il risultato finale della valutazione possa essere determinato dopo aver valutato solo una parte dell'espressione. Ad esempio, se si valuta la decisione "A AND B", non è necessario valutare B se A è già stato valutato come FALSO. Nessun valore di B può modificare il risultato finale, quindi il codice può far risparmiare i tempi di esecuzione non valutando B. L’uso del cortocircuito può influenzare la capacità di raggiungere MC/DC poiché alcuni test richiesti possono non essere realizzabili. Generalmente, è possibile configurare il compilatore per non eseguire l’ottimizzazione del cortocircuito durante il testing, ma questo non è permesso per le applicazioni safety_critical, dove il codice testato e il codice rilasciato devono essere identici

2.5 Testing delle Condizioni Multiple

In rari casi, potrebbe essere richiesto di testare tutte le possibili combinazioni delle condizioni atomiche che una decisione può contenere. Questo livello di test è chiamato testing delle condizioni multiple. Assumendo N condizioni atomiche uniche e mutualmente indipendenti, la copertura completa delle condizioni multiple per una decisione può essere raggiunta esercitando la decisione per 2^N volte. Si osservi che un singolo test case può esercitare diverse combinazioni di condizioni e quindi non è sempre necessario eseguire 2^N test case separati per raggiungere il 100%di copertura delle condizioni multiple.

La copertura viene misurata come numero di combinazioni di condizioni atomiche esercitate rispetto a tutte le decisioni nell'oggetto di test, normalmente espressa come percentuale.

Applicabilità

Questa tecnica viene utilizzata per testare software ad alto rischio e software embedded che ci si aspetta che funzioni in modo affidabile, senza generare crash per lunghi periodi di tempo.

Limitazioni/Difficoltà

Poiché il numero di test case può essere derivato direttamente da una tabella delle verità che contiene tutte le condizioni atomiche, questo livello di copertura può essere facilmente determinato. Tuttavia, l'enorme numero di test case richiesti per il testing delle condizioni multiple rende più fattibile il testing delle decisioni/condizioni modificate, nel caso in cui esistano diverse condizioni atomiche in una decisione.

Se il compilatore utilizza cortocircuiti (short-circuiting), il numero di combinazioni delle condizioni che possono essere esercitate sarà spesso ridotto, in base all'ordine e al raggruppamento delle operazioni logiche che vengono eseguite sulle condizioni atomiche.

2.6 Testing dei Cammini Base

Questo capitolo è stato eliminato dalla versione v4.0 di questo Syllabus

2.7 Testing delle API

Una API (Application Programming Interface) è un'interfaccia definita che permette ad un programma di chiamare un altro sistema software che gli fornisce un servizio, come l'accesso a una risorsa remota. I servizi tipici includono servizi Web, enterprise service bus, database, mainframe e interfacce utente (UI, User Interface) Web.

Il testing delle API è un tipo di test piuttosto che una tecnica. Per certi aspetti, il testing delle API è abbastanza similare al testing di un'interfaccia GUI (Graphical User Interface). Il focus è sulla valutazione dei valori di input e dei dati di output.

Quando si tratta di API il testing negativo è spesso cruciale. I programmatori che utilizzano le API per permettere a servizi esterni di accedere al proprio codice, possono cercare di utilizzare le interfacce API nei modi per cui non erano previste. Questo significa che è necessaria una gestione degli errori robusta, per evitare operazioni errate. Può essere richiesto di eseguire il testing combinatorio di molte interfacce, perché le API sono spesso utilizzate insieme ad altre API, e perché una singola interfaccia può contenere diversi parametri, dove i valori di questi possono essere combinati in modi differenti.

Spesso le API sono debolmente accoppiate, e questo può risultare nella possibilità molto reale di transazioni perse o problemi di temporizzazione. Questo richiede un testing approfondito dei meccanismi di recovery e retry. Un'organizzazione che fornisce un'interfaccia API deve garantire che tutti i servizi abbiano una disponibilità molto elevata; questo spesso richiede un testing di affidabilità rigoroso da parte di chi rilascerà la API, oltre a un supporto infrastrutturale.

Applicabilità

Il testing delle API sta diventando sempre più importante per il testing di sistemi di sistemi, man mano che i singoli sistemi diventano distribuiti o utilizzano l'elaborazione remota (remote processing) come metodo per scaricare il carico di lavoro su altri processori. Alcuni esempi includono:

- Chiamate ai sistemi operativi
- Service-Oriented architecture (SOA)
- Remote Procedure Call (RPC)
- Web service

L'uso di container software si traduce nella suddivisione di un programma software in diversi container che comunicano tra loro utilizzando meccanismi come quelli elencati sopra. Il testing delle API dovrebbe anche indirizzare queste interfacce.

Limitazioni/Difficoltà

Il testing direttamente su una API di solito richiede che un Technical Test Analyst utilizzi strumenti specializzati. Poiché in genere non esiste un'interfaccia grafica diretta associata a una API, possono essere richiesti strumenti per configurare l'ambiente iniziale, organizzare (marshall) i dati, invocare le API e determinare il risultato.

Copertura

Il testing delle API è una descrizione di un tipo di test e non definisce alcun livello specifico di copertura. Come minimo, il testing delle API dovrebbe includere le chiamate alle API con valori di input realistici e con input non validi per verificare la gestione delle eccezioni. Test delle API più approfonditi possono assicurare che le entità chiamabili vengano esercitate almeno una volta, o che tutte le possibili funzioni vengano chiamate almeno una volta.

Representational State Transfer è uno stile architetturale. I servizi Web RESTful permettono ai sistemi richiedenti di accedere a risorse Web utilizzando un insieme uniforme di operazioni senza stato. Per le API Web RESTful esistono diversi criteri di copertura, che sono lo standard di fatto per l'integrazione del software. [Web-7]. Possono essere suddivisi in due gruppi: criteri di copertura di input e criteri di copertura di output. Tra gli altri, i criteri di input possono richiedere l'esecuzione di tutte le possibili operazioni delle API, l'uso di tutti i possibili parametri delle API, e la copertura delle sequenze di operazioni delle API. Tra gli altri, i criteri di output possono richiedere la generazione di tutti i codici di stato corretti ed errati, e la generazione di risposte che contengono le risorse che esibiscono tutte le proprietà (o tutti i tipi di proprietà).

Tipi di difetti

Esistono diversi tipi di difetti che possono essere rilevati durante il testing delle API. Sono comuni i problemi di interfaccia, come i problemi nella gestione dei dati, i problemi di temporizzazione, la perdita delle transazioni, la duplicazione di transazioni e i problemi nella gestione delle eccezioni.

2.8 Selezionare una Tecnica di Test White-Box

La tecnica di test white-box selezionata è normalmente specificata in termini di livello di copertura richiesto, che viene raggiunto applicando la tecnica di test. Ad esempio, un requisito per ottenere una copertura del 100% delle istruzioni porterebbe in genere ad utilizzare il testing delle istruzioni. Comunque, le tecniche di test black-box vengono normalmente applicate per prime, viene quindi misurata la copertura, e la tecnica di test white-box viene applicata solo se il livello di copertura white-box richiesto non è stato raggiunto. In alcune situazioni, il testing white-box può essere utilizzato in modo meno formale per fornire un'indicazione di dove potrebbe essere necessario aumentare la copertura (ad es. creando test aggiuntivi dove i livelli di copertura white-box sono particolarmente bassi). Il testing delle istruzioni sarebbe normalmente sufficiente per tale misurazione informale della copertura.

Quando si specifica un livello di copertura white-box richiesto, è buona pratica specificarlo solo al 100%. La ragione di questo è che se sono richiesti livelli di copertura inferiori, questo in genere significa che le

parti del codice che non vengono esercitate dal testing sono le parti più difficili da testare, e queste parti sono normalmente anche le più complesse e soggette a errori. Quindi, richiedendo e raggiungendo ad esempio una copertura dell'80%, questo può significare che il codice che include la maggior parte dei difetti rilevabili non viene testato. Per questo motivo, quando i criteri di copertura white-box vengono specificati negli standard, sono quasi sempre specificati al 100%. Definizioni rigorose dei livelli di copertura hanno talvolta reso questo livello di copertura irraggiungibile. Tuttavia, quelli forniti nello standard ISO 29119-4 consentono di escludere dai calcoli gli elementi di copertura non fattibili, rendendo così la copertura del 100% un obiettivo raggiungibile.

Quando si specifica la copertura white-box richiesta per un oggetto di test, è anche necessario specificarla solo per un singolo criterio di copertura (ad es. non è necessario richiedere sia la copertura del 100% delle istruzioni sia il 100% MC/DC). Con i criteri di uscita al 100% è possibile correlare alcuni criteri di uscita attraverso una gerarchia di correlazioni, dove i criteri di copertura sono correlati a criteri di copertura più ampi. Si dice che un criterio di copertura ne include (o sussume) un altro se, per tutti i componenti e le loro specifiche, ogni insieme di test case che soddisfa il primo criterio soddisfa anche il secondo. Ad esempio, la copertura dei rami include la copertura delle istruzioni perché se viene raggiunta la copertura dei rami (al 100%), viene anche garantita la copertura delle istruzioni al 100%. Per le tecniche di test white-box coperte in questo Syllabus, la copertura dei rami e delle decisioni include la copertura delle istruzioni, MC/DC include la copertura delle decisioni e dei rami, e la copertura delle condizioni multiple include MC/DC (se si considera la copertura dei rami e delle decisioni al 100%, allora le coperture si includono a vicenda).

Si osservi che quando si determinano i livelli di copertura white-box che un sistema deve raggiungere, è normale definire livelli differenti per parti differenti del sistema. Questo perché diverse parti di un sistema contribuiscono in modo diverso al rischio. Ad esempio, in un sistema avionico, ai sottosistemi associati all'intrattenimento in volo verrebbe assegnato un livello di rischio inferiore rispetto a quello associato al sottosistema di controllo del volo. Il testing delle interfacce è comune per tutti i tipi di sistemi ed è normalmente richiesto per tutti i livelli di integrità (integrity level) dei sistemi safety-critical (si veda il paragrafo 2.7.2 per ulteriori informazioni sui livelli di integrità). Normalmente il livello di copertura richiesto per il testing delle API aumenterà in base al rischio associato (ad es. il livello di rischio più alto associato a un'interfaccia pubblica può richiedere un testing delle API più rigoroso).

La selezione della tecnica di test white-box da utilizzare si basa generalmente sulla natura dell'oggetto di test e sui relativi rischi percepiti. Se l'oggetto di test viene considerato safety (ovvero un failure potrebbe causare danni alle persone o all'ambiente), sono applicabili standard normativi che definiranno i livelli di copertura white-box richiesti (si veda il paragrafo 2.7.2). Se l'oggetto di test non è safety, la scelta dei livelli di copertura white-box da raggiungere è più soggettiva, ma dovrebbe anche essere ampiamente basata sui rischi percepiti, come descritto nel paragrafo 2.7.1.

2.8.1 Sistemi Non-Safety

Quando si selezionano le tecniche di test white-box per sistemi non-safety, sono normalmente considerati i seguenti fattori (non in un particolare ordine di priorità):

- Contratto - Se il contratto richiede che venga raggiunto un particolare livello di copertura, il mancato raggiungimento di questo livello di copertura genererà potenzialmente una violazione del contratto.
- Cliente - Se il cliente richiede un particolare livello di copertura, ad esempio come parte della pianificazione dei test, il mancato raggiungimento di questo livello di copertura può causare problemi con il cliente.
- Standard normativi - Per alcuni settori industriali (ad es. finanziario) si applica uno standard normativo che definisce i criteri di copertura white-box richiesti per i sistemi mission-critical. Vedere il paragrafo 2.7.2 per la copertura degli standard normativi per sistemi safety.

- Strategia di test - Se la strategia di test dell'organizzazione specifica i requisiti per la copertura del codice white-box, il mancato allineamento con la strategia dell'organizzazione può rischiare la censura da parte del management dei livelli più alti.
- Stile di codifica - Se il codice viene scritto senza condizioni multiple all'interno delle decisioni, sarebbe uno spreco richiedere livelli di copertura white-box come MC/DC e copertura delle condizioni multiple.
- Informazioni storiche sui difetti - Se i dati storici sull'efficacia del raggiungimento di un particolare livello di copertura suggeriscono che sarebbe appropriato utilizzarli per questo oggetto di test, sarebbe rischioso ignorare i dati disponibili. Si noti che tali dati possono essere disponibili all'interno del progetto, dell'organizzazione o del settore industriale.
- Competenza ed esperienza – Se i tester disponibili per eseguire il testing non sono sufficientemente esperti e competenti in una particolare tecnica white-box, può essere non compresa e possono essere introdotti rischi inutili se tale tecnica è stata selezionata.
- Strumenti - La copertura white-box può essere misurata nella pratica solo utilizzando strumenti di copertura. Se tali strumenti non sono disponibili a supporto di una determinata misura di copertura, la selezione della misura da raggiungere introdurrebbe un livello di rischio elevato.

Quando si seleziona il testing white-box per sistemi non-safety, il Technical Test Analyst ha maggiore libertà di raccomandare la copertura white-box appropriata per i sistemi non-safety rispetto ai sistemi safety. Tali scelte sono in genere un compromesso tra i rischi percepiti e i costi, le risorse e i tempi richiesti per gestire questi rischi attraverso il testing white-box. In alcune situazioni, possono essere più appropriati altri trattamenti, che potrebbero essere implementati da altri approcci del testing del software oppure in un altro modo (ad es. da diversi approcci di sviluppo).

2.8.2 Sistemi Safety

Quando il software da testare è parte di un sistema safety, normalmente dovrà essere utilizzato uno standard normativo che definisce i livelli di copertura richiesti da raggiungere. Tali standard richiedono in genere l'esecuzione di un'analisi degli hazard per il sistema, e i rischi risultanti vengono utilizzati per assegnare livelli di integrità alle diverse parti del sistema. I livelli di copertura richiesti sono definiti per ciascun livello di integrità.

IEC 61508 (safety funzionale di sistemi programmabili, elettronici e safety [IEC 61508]) è uno standard internazionale generico utilizzato per tali scopi. In teoria, potrebbe essere utilizzato per qualsiasi sistema safety, tuttavia alcune industrie hanno creato varianti specifiche (ad es. ISO 26262 [ISO 26262] si applica ai sistemi automotive) e alcune industrie hanno creato i propri standard (ad es. DO-178C [DO-178C] per software avionico). Ulteriori informazioni relative a ISO 26262 sono fornite nel Syllabus ISTQB® Automotive Software Tester [CTSL_AuT_SYL].

IEC 61508 definisce quattro safety integrity level (livelli di integrità safety, SIL), ognuno dei quali viene definito come un livello relativo di riduzione del rischio fornito da una funzione safety, correlato alla frequenza e alla severità degli hazard percepiti. Quando un oggetto di test esegue una funzione correlata a safety, un maggior rischio di failure significa che l'oggetto di test dovrebbe avere maggiore affidabilità. La tabella seguente mostra i livelli di affidabilità associati ai SIL. Si noti che il livello di affidabilità per SIL 4 per il caso di continuous operation (funzionamento continuo) è estremamente elevato, poiché corrisponde a un Mean Time Between Failures (MTBF) maggiore di 10.000 anni.

SIL IEC 61508	Continuous Operation (probabilità di un failure dannoso all'ora)	On-Demand (probabilità di failure on demand)
1	$\geq 10^{-6}$ to $<10^{-5}$	$\geq 10^{-2}$ to $<10^{-1}$

2	$\geq 10^{-7}$ to $<10^{-6}$	$\geq 10^{-3}$ to $<10^{-2}$
3	$\geq 10^{-8}$ to $<10^{-7}$	$\geq 10^{-4}$ to $<10^{-3}$
4	$\geq 10^{-9}$ to $<10^{-8}$	$\geq 10^{-5}$ to $<10^{-4}$

Le raccomandazioni per i livelli di copertura white-box associati ad ogni SIL sono rappresentate nella seguente tabella. Dove una voce della tabella viene indicata come "Altamente raccomandato", in pratica il raggiungimento di tale livello di copertura è in genere considerato obbligatorio. Al contrario, quando una voce della tabella viene indicata come "Consigliata", molti professionisti considerano opzionale il raggiungimento di tale livello di copertura ed evitano di raggiungerla fornendo una motivazione adeguata. Pertanto, un oggetto di test assegnato al livello SIL 3 viene normalmente testato per ottenere il 100% di copertura dei rami (e quindi automaticamente il 100% di copertura delle istruzioni, come indicato dall'ordine delle inclusioni delle coperture).

SIL IEC 61508	**100% di Copertura delle istruzioni**	**100% di Copertura dei rami**	**100% MC/DC**
1	Raccomandato	Raccomandato	Raccomandato
2	Altamente raccomandato	Raccomandato	Raccomandato
3	Altamente raccomandato	Altamente raccomandato	Raccomandato
4	Altamente raccomandato	Altamente raccomandato	Altamente raccomandato

Si noti che i SIL e i requisiti sui livelli di copertura dello standard IEC 61508 sono differenti da quelli presenti in ISO 26262 e DO-178-C.

3. Analisi Statica e Dinamica - 180 minuti

Parole Chiave

analisi del flusso dati, analisi del flusso di controllo, analisi dinamica, analisi statica, associazione definizione-utilizzo, complessità ciclomatica, memory leak, puntatore errato

Obiettivi di Apprendimento per l'Analisi Statica e Dinamica

3.2 Analisi Statica

TTA-3.2.1 (K3) Utilizzare l'analisi del flusso di controllo per rilevare se il codice presenta anomalie nel flusso di controllo e per misurare la complessità ciclomatica
TTA-3.2.2 (K3) Utilizzare l'analisi del flusso dati per rilevare se il codice presenta anomalie nel flusso dati
TTA-3.2.3 (K3) Proporre modi per migliorare la manutenibilità del codice applicando l'analisi statica

Nota: TTA-3.2.4 è stato eliminato dalla versione V2.0 di questo Syllabus.

3.3 Analisi Dinamica

TTA-3.3.1 (K3) Applicare l'analisi dinamica per raggiungere un obiettivo specifico

3.1 Introduzione

L'analisi statica (si veda il Paragrafo 3.2) è una forma di testing che viene svolto senza eseguire il software. La qualità del software è valutata da uno strumento o da una persona in base al suo formato, alla sua struttura, al suo contenuto o alla sua documentazione. Questa vista statica del software consente un'analisi dettagliata senza dover creare i dati e le precondizioni necessari per eseguire i test case.

Oltre all'analisi statica, le tecniche di test statico includono anche diverse forme di review. Le review che sono rilevanti per il Technical Test Analyst sono trattate nel Capitolo 5.

L'analisi dinamica (si veda il Paragrafo 3.3) richiede l'esecuzione effettiva del codice e viene utilizzata per rilevare difetti che sono rilevati più facilmente quando il codice è in esecuzione (ad es. memory leak). L'analisi dinamica, come l'analisi statica, può fare affidamento su strumenti o su una persona che monitora il sistema in esecuzione osservando indicatori come un rapido aumento nell'uso della memoria.

3.2 Analisi Statica

L'obiettivo dell'analisi statica è rilevare difetti reali o potenziali nel codice e nell'architettura del sistema e migliorarne la manutenibilità.

3.2.1 Analisi del Flusso di Controllo

L'analisi del flusso di controllo è la tecnica statica dove vengono analizzati i passi seguiti da un programma, attraverso l'uso di un grafo del flusso di controllo, normalmente utilizzando uno strumento. Esistono un certo numero di anomalie che possono essere rilevate in un sistema utilizzando questa tecnica, inclusi cicli progettati in modo errato (ad es. con punti di ingresso multipli o che non terminano), chiamate di funzione ambigue in alcuni linguaggi (ad es. Scheme), sequenza errata delle operazioni, codice irraggiungibile, funzioni non chiamate, ecc.

L'analisi del flusso di controllo può essere utilizzata per determinare la complessità ciclomatica. La complessità ciclomatica è un numero intero positivo che rappresenta il numero di cammini indipendenti in un grafo fortemente connesso.

La complessità ciclomatica viene generalmente utilizzata come indicatore della complessità di un componente. La teoria di Thomas McCabe [McCabe 76] sosteneva che più complesso è il sistema, più sarebbe difficile manutenerlo e più difetti sarebbero presenti in tale sistema. Molti studi hanno rilevato questo tipo di correlazione tra complessità ciclomatica e numero di difetti presenti. Qualsiasi componente con una complessità maggiore dovrebbe essere oggetto di review, per un possibile refactoring, ad esempio la suddivisione in più componenti.

3.2.2 Analisi del Flusso Dati

L'analisi del flusso dati copre una varietà di tecniche che raccolgono informazioni sull'utilizzo delle variabili in un sistema. Viene analizzato il ciclo di vita di ogni variabile nel cammino del flusso di controllo (cioè dove viene dichiarata, definita, usata e cancellata), poiché possono essere identificate potenziali anomalie se queste azioni vengono utilizzate fuori sequenza [Beizer90].

Una tecnica comune classifica l'uso di una variabile come una delle tre azioni atomiche:

- Quando la variabile viene definita, dichiarata, o inizializzata (ad es. x:=3)
- Quando la variabile viene utilizzata (ad es. x > temp)

- Quando la variabile viene eliminata, cancellata, o esce dall'ambito (out of scope) (ad es. text_file_1.close, loop control variable (i) on exit from loop)

Sequenze di tali azioni che indicano potenziali anomalie includono:

- Definizione seguita da un'altra definizione o cancellazione senza alcun utilizzo intermedio
- Definizione senza cancellazione successiva (ad.es possibili memory leak per variabili allocate dinamicamente)
- Utilizzo o cancellazione prima della definizione
- Utilizzo o cancellazione dopo una cancellazione

In base al linguaggio di programmazione, alcune di queste anomalie possono essere identificate dal compilatore, ma potrebbe essere necessario uno strumento di analisi statica separato per identificare le anomalie del flusso dati. Ad esempio, la ridefinizione senza utilizzo intermedio è consentita nella maggior parte dei linguaggi di programmazione e può essere programmata deliberatamente, ma sarebbe segnalata da uno strumento di analisi del flusso dati come una possibile anomalia che dovrebbe essere verificata.

L'utilizzo dei cammini del flusso di controllo per determinare la sequenza di azioni per una variabile può portare alla segnalazione di potenziali anomalie che nella pratica non possono verificarsi. Ad esempio, gli strumenti di analisi statica non possono sempre identificare se un cammino del flusso di controllo è fattibile, poiché alcuni cammini sono determinati solo in base ai valori assegnati alle variabili in fase di esecuzione. Esiste anche una classe di problemi di analisi del flusso dati che sono difficili da identificare da parte degli strumenti, quando i dati analizzati sono parte di strutture dati con variabili assegnate dinamicamente, come record e array. Gli strumenti di analisi statica possono anche rendere difficile l'identificazione di potenziali anomalie del flusso dati quando le variabili sono condivise tra thread concorrenti di controllo in un programma, poiché la sequenza di azioni sui dati diventa difficile da prevedere.

A differenza dell'analisi del flusso dati, che è un testing statico, il testing del flusso dati è un testing dinamico in cui vengono generati i test case per esercitare " associazione definizione-utilizzo" nel codice del programma. Il testing del flusso dati utilizza alcuni dei concetti utilizzati nell'analisi del flusso dati, poiché queste associazioni definizione-utilizzo sono cammini del flusso di controllo tra una definizione e un successivo utilizzo di una variabile in un programma.

3.2.3 Utilizzo dell'Analisi Statica per Migliorare la Manutenibilità

L'analisi statica può essere applicata in diversi modi per migliorare la manutenibilità del codice, dell'architettura e dei siti web.

Il codice scritto male, non commentato e non strutturato tende ad essere più difficile da manutenere. Può richiedere agli sviluppatori un effort maggiore per localizzare e analizzare i difetti nel codice, e la modifica del codice per correggere un difetto o per aggiungere una nuova funzionalità comporta probabilmente l'introduzione di ulteriori difetti.

L'analisi statica è utilizzata per verificare la conformità agli standard e alle linee guida di codifica; dove viene identificato codice non conforme, il codice può essere aggiornato per migliorare la sua manutenibilità. Questi standard e linee guida descrivono pratiche di progettazione e codifica richieste quali naming convention, uso di commenti, indentazione e modularizzazione. Si noti che gli strumenti di analisi statica generalmente producono warning (segnalazioni) piuttosto che rilevare errori. Questi warning (ad es. sul livello di complessità) possono essere evidenziati anche se il codice è sintatticamente corretto.

La progettazione modulare generalmente produce un codice più manutenibile. Gli strumenti di analisi statica supportano lo sviluppo di codice modulare nei seguenti modi:

- Cercano codice ripetuto. Queste parti di codice possono essere candidate per un refactoring nei componenti (sebbene il runtime overhead imposto dalle chiamate dei componenti può essere un problema per i sistemi real-time).
- Generano metriche che sono indicatori preziosi della modularizzazione del codice. Questi includono misure di accoppiamento (coupling) e coesione (cohesion). Un sistema che ha una buona manutenibilità ha maggiore probabilità di avere una bassa misura di accoppiamento (il grado con cui i componenti dipendono uno con l'altro durante l'esecuzione) e un'alta misura di coesione (il grado con cui un componente è completo e autonomo, e focalizzato su una singola attività).
- Identificano in un codice object-oriented dove gli oggetti figli possono avere troppa o troppo poca visibilità nelle classi parent.
- Evidenziano aree nel codice o nell'architettura con un alto livello di complessità strutturale.

Anche la manutenzione di un sito web può essere supportata dall'utilizzo di strumenti di analisi statica. In questo caso l'obiettivo è verificare se la struttura ad albero del sito è ben bilanciata, oppure esiste uno sbilanciamento che porterà a:

- Attività di test più difficili
- Aumento del carico di lavoro di manutenzione

3.3 Analisi Dinamica

3.3.1 Panoramica

L'analisi dinamica viene utilizzata per rilevare failure dove i sintomi sono visibili solo quando il codice viene eseguito. Ad esempio, la presenza di memory leak può essere rilevabile dall'analisi statica (rilevare codice che alloca ma non libera mai memoria), ma è immediatamente evidente con l'analisi dinamica.

I failure che non sono immediatamente riproducibili (intermittenti) possono avere conseguenze significative sull'effort del testing e sulla capacità di rilasciare o utilizzare in modo produttivo il software. Tali failure possono essere causati da memory leak e resource leak, da un uso non corretto dei puntatori o da altri strutture corrotte (ad es. stack di sistema corrotto) [Kaner02]. A causa della natura di questi failure, che possono includere il graduale peggioramento delle prestazioni del sistema o addirittura crash di sistema, le strategie di test devono considerare i rischi associati a tali difetti e, quando appropriato, eseguire l'analisi dinamica per ridurli (tipicamente utilizzando degli strumenti). Poiché questi failure sono spesso i più costosi da rilevare e correggere, è raccomandato di iniziare l'analisi dinamica nelle fasi inziali di progetto.

L'analisi dinamica può essere applicata per ottenere quanto segue:

- Prevenire il verificarsi di failure rilevando memory leak (si veda il paragrafo 3.3.2) e puntatori errati (si veda il paragrafo 3.3.3)
- Analizzare failure di sistema che non possono essere facilmente riprodotti
- Valutare il comportamento della rete
- Migliorare le prestazioni del sistema utilizzando code profiler, per fornire informazioni sul comportamento del sistema in esecuzione che possano essere utilizzate per apportare modifiche ragionate.

L'analisi dinamica può essere eseguita a qualsiasi livello di test e richiede competenze tecniche e di sistema per eseguire le seguenti operazioni:

- Specificare gli obiettivi di test dell'analisi dinamica

- Determinare il momento più adatto per avviare e interrompere l'analisi
- Analizzare i risultati.

E' possibile utilizzare strumenti di analisi dinamica anche se il Technical Test Analyst ha competenze tecniche minime; gli strumenti utilizzati solitamente creano log completi che possono essere analizzati da coloro che possiedono le competenze tecniche e analitiche richieste.

3.3.2 Rilevare Memory Leak

I memory leak si verificano quando aree di memoria (RAM) vengono allocate a un programma ma non vengono successivamente rilasciate quando non più necessarie. Queste aree di memoria rimangono non disponibili per un riutilizzo. Quando questo si verifica frequentemente o in situazioni di memoria insufficiente, il programma può esaurire la memoria utilizzabile. Storicamente, la gestione della memoria era sotto la responsabilità del programmatore. Qualsiasi area di memoria allocata dinamicamente doveva essere rilasciata dal programma che la allocava per evitare un memory leak. Molti ambienti di programmazione moderni includono un sistema automatico o semiautomatico di garbage collection in cui la memoria allocata viene liberata dopo l'uso senza l'intervento diretto del programmatore. Isolare i memory leak nei casi in cui la memoria allocata dovrebbe essere liberata dai Garbage Collection automatici può essere molto difficile.

I memory leak causano generalmente problemi dopo qualche tempo, quando una quantità di memory leak è significativa e la memoria diventa non disponibile. Un esempio di allocazione di memoria frequente che può prevenire il rilevamento di memory leak è quando il software è stato installato di recente o quando il sistema è stato riavviato, e in questi casi la memoria viene riallocata e i memory leak non sono rilevabili. Per questi motivi, gli effetti negativi dei memory leak possono essere rilevati per la prima volta quando il programma è in produzione.

Il sintomo principale di un memory leak è un costante peggioramento dei tempi di risposta del sistema, che alla fine può provocare un failure di sistema. Sebbene tali failure possano essere risolti riavviando il sistema (reboot), questo può non essere sempre pratico o addirittura possibile per alcuni sistemi.

Molti strumenti di analisi dinamica identificano le aree del codice in cui si verificano memory leak in modo che tali leak possano essere corretti. Si possono usare anche semplici monitor di memoria per ottenere un'indicazione generale della diminuzione della memoria disponibile nel tempo, sebbene sia comunque necessaria un'analisi successiva per determinare la causa esatta della diminuzione.

3.3.3 Rilevare Puntatori Errati

I puntatori errati (wild pointer) all'interno di un programma sono puntatori che non sono più accurati e che non devono essere utilizzati. Ad esempio, un puntatore errato potrebbe aver "perso" l'oggetto o la funzione a cui dovrebbe puntare, oppure potrebbe non puntare all'area di memoria prevista (ad es. punta a un'area che è oltre i limiti allocati di un array). Quando un programma utilizza puntatori errati possono verificarsi diverse conseguenze, tra cui:

- Il programma può funzionare come previsto. Questo può essere il caso in cui il puntatore errato accede a memoria che non è attualmente utilizzata ed è teoricamente "libera" e/o contiene un valore ragionevole.
- Il programma può generare un crash. Questo può essere il caso in cui il puntatore errato ha causato un uso non corretto di una parte della memoria che diventa critico per l'esecuzione del programma (ad es. il sistema operativo).
- Il programma non funziona correttamente perché non è possibile da parte del programma accedere agli oggetti richiesti. In queste condizioni il programma può continuare a funzionare, magari generando un messaggio di errore.

- I dati nella memoria possono essere corrotti dal puntatore e vengono successivamente utilizzati (questo può anche rappresentare un problema di sicurezza).

Si noti che modifiche all'utilizzo della memoria del programma (ad es. una nuova build a seguito di una modifica del software) può essere il trigger di una delle quattro conseguenze sopra descritte. Questo è particolarmente critico quando inizialmente il programma funziona come previsto, nonostante l'uso di puntatori errati, e poi genera un crash imprevisto (anche in produzione) a seguito di una modifica del software. Esistono strumenti che possono aiutare a identificare puntatori errati, indipendentemente dal loro impatto sull'esecuzione del programma. Alcuni sistemi operativi possiedono funzioni integrate per verificare durante l'esecuzione le violazioni di accesso alla memoria. Ad esempio, il sistema operativo può generare un'eccezione quando un'applicazione tenta di accedere a una posizione di memoria che è al di fuori dell'area di memoria consentita di quell'applicazione.

3.3.4 Analisi di Efficienza delle Prestazioni

L'analisi dinamica non è utile solo per rilevare i failure e localizzare i difetti associati. Con l'analisi dinamica delle prestazioni del programma, gli strumenti aiutano a identificare colli di bottiglia per l'efficienza delle prestazioni e generare un'ampia gamma di metriche delle prestazioni che possono essere utilizzate dallo sviluppatore per ottimizzare le prestazioni del sistema. Ad esempio, è possibile fornire informazioni sul numero di volte in cui un componente viene chiamato durante l'esecuzione. I componenti che vengono chiamati frequentemente sono i probabili candidati per un'attività di miglioramento delle prestazioni. Spesso vale la regola di Pareto: un programma spende una parte sproporzionata (80%) del suo tempo di esecuzione in un piccolo numero (20%) di componenti [Andrist20].
L'analisi dinamica delle prestazioni del programma viene spesso eseguita durante il testing di sistema, ma può anche essere eseguita durante il testing di un singolo sottosistema nelle fasi precedenti di test, utilizzando test harness. Ulteriori dettagli sono riportati nel Syllabus Performance Testing [CTSL_PT_SYL].

4. Caratteristiche di Qualità per il Testing Tecnico - 345 minuti

Parole Chiave
adattabilità, affidabilità, analizzabilità, autenticità, capacità, caratteristica di qualità, coesistenza, compatibilità, comportamento nel tempo, confidenzialità, disponibilità, efficienza delle prestazioni, installabilità, integrità, manutenibilità, maturità, modello di crescita dell'affidabilità, modificabilità, modularità, non-repudiation, profilo operativo, portabilità, recuperabilità responsabilità (accountability), riutilizzabilità, sicurezza, sostituibilità, testabilità, tolleranza ai guasti, utilizzo delle risorse

Obiettivi di Apprendimento per le Caratteristiche di Qualità per il Testing Tecnico

4.2 Aspetti Generali di Pianificazione

TTA-4.2.1 (K4) Per un particolare scenario, analizzare i requisiti non funzionali e scrivere i rispettivi paragrafi del Test Plan
TTA-4.2.2 (K3) Dato un particolare rischio di prodotto, definire i tipi di test non funzionali più appropriati
TTA-4.2.3 (K2) Comprendere e spiegare le fasi in un ciclo di vita di sviluppo software di un'applicazione in cui il testing non funzionale dovrebbe essere normalmente applicato
TTA-4.2.4 (K3) In un dato scenario, definire i tipi di difetti che ci si aspetterebbe di rilevare utilizzando i diversi tipi di test non funzionali

4.3 Testing di Sicurezza

TTA-4.3.1 (K2) Spiegare le ragioni per cui includere il testing di sicurezza in un approccio di test
TTA-4.3.2 (K2) Spiegare gli aspetti principali da prendere in considerazione nella pianificazione e nella specifica dei test di sicurezza

4.4 Testing di Affidabilità

TTA-4.4.1 (K2) Spiegare le ragioni per cui includere il testing di affidabilità in un approccio di test
TTA-4.4.2 (K2) Spiegare gli aspetti principali da prendere in considerazione nella pianificazione e nella specifica dei test di affidabilità

4.5 Performance Testing (Testing di Efficienza delle Prestazioni)

TTA-4.5.1 (K2) Spiegare le ragioni per cui includere il performance testing in un approccio di test
TTA-4.5.2 (K2) Spiegare gli aspetti principali da prendere in considerazione nella pianificazione e nella specifica del performance testing

4.6 Testing di Manutenibilità

TTA-4.6.1 (K2) Spiegare le ragioni per cui includere il testing di manutenibilità in un approccio di test

4.7 Testing di Portabilità

TTA-4.7.1 (K2) Spiegare le ragioni per cui includere il testing di portabilità in un approccio di test

4.8 Testing di Compatibilità

TTA-4.8.1 (K2) Spiegare le ragioni per cui includere il testing di coesistenza in un approccio di test

4.1 Introduzione

In generale, il Technical Test Analyst si focalizza sul testing di "come" funziona il prodotto, piuttosto che sul testing degli aspetti funzionali di "cosa" fa il prodotto. Questi test possono essere svolti a qualsiasi livello di test. Ad esempio, durante il testing di componente di sistemi embedded e real-time è importante condurre un'analisi comparativa sull'efficienza delle prestazioni e testare l'utilizzo delle risorse. Durante il testing di accettazione operativa e il testing di sistema, è appropriato eseguire il testing degli aspetti di affidabilità, come la recuperabilità. I test a questo livello hanno lo scopo di testare un sistema specifico, cioè una combinazione di hardware e software. Il sistema specifico sotto test può includere diversi server, client, database, reti e altre risorse. Indipendentemente dal livello di test, il testing dovrebbe essere condotto in base alle priorità di rischio e alle risorse disponibili.

Sia il testing dinamico che il testing statico, incluse le review (si vedano i Capitoli 2,3 e 5) possono essere applicati al testing delle caratteristiche di qualità non funzionali.

La descrizione delle caratteristiche e sotto-caratteristiche di qualità del prodotto fa riferimento allo standard ISO 25010 [ISO25010]. La tabella seguente riporta anche un'indicazione di quali caratteristiche/sotto-caratteristiche sono coperte dai Syllabi Test Analyst e Technical Test Analyst.

Caratteristica	Sotto-Caratteristica	Test Analyst	Technical Test Analyst
Adeguatezza funzionale	Correttezza funzionale, appropriatezza funzionale, completezza funzionale	X	
Affidabilità	Maturità, tolleranza ai guasti, recuperabilità, disponibilità		X
Usabilità	Riconoscibilità dell'appropriatezza, apprendibilità, operabilità, estetica dell'interfaccia utente, protezione da errori dell'utente, accessibilità	X	
Efficienza delle prestazioni	Comportamento nel tempo, utilizzo delle risorse, capacità		X
Manutenibilità	Analizzabilità, modificabilità, testabilità, modularità, riutilizzabilità		X
Portabilità	Adattabilità, installabilità, sostituibilità		X
Sicurezza	Confidenzialità, integrità, non-repudiation, responsabilità, autenticità		X
Compatibilità	Coesistenza		X
	Interoperabilità	X	

Si noti che l'Appendice A riporta una tabella che confronta le caratteristiche descritte nello standard ora cancellato ISO 9126-1 (utilizzato nella versione 2012 di questo Syllabus) con quelle nello standard più recente ISO 25010.

Per tutte le caratteristiche e le sotto-caratteristiche di qualità discusse in questo paragrafo, occorre riconoscere i rischi tipici, in modo che possa essere definito e documentato un appropriato approccio di test. Il testing delle caratteristiche di qualità richiede un'attenzione particolare alla tempistica del ciclo di vita, agli strumenti richiesti, agli standard richiesti, alla disponibilità del software e della documentazione, e all'esperienza tecnica. Senza la pianificazione di un approccio che tenga conto di ogni caratteristica e della relativa necessità specifica di testing, la schedulazione può essere definita non fornendo al tester il tempo adeguato per la pianificazione, la preparazione e l'esecuzione dei test.

Alcuni di questi tipi di test, ad esempio il performance testing, richiedono una pianificazione dettagliata, attrezzature dedicate, strumenti specifici, competenze di test specializzate e, nella maggior parte dei casi, una notevole quantità di tempo. Il testing delle caratteristiche e delle sotto-caratteristiche di qualità

deve essere integrato nella schedulazione complessiva di test, con adeguate risorse allocate a tale sforzo.

La compilazione e il reporting delle informazioni riassuntive delle metriche relative alle caratteristiche e alle sotto-caratteristiche di qualità sono attività svolte dal Test Manager, mentre il Test Analyst o il Technical Test Analyst (in base alla tabella sopra riportata) hanno il compito di raccogliere le informazioni di ciascuna metrica.

Le metriche delle caratteristiche di qualità raccolte nei test di pre-produzione dal Technical Test Analyst possono costituire la base dei Service Level Agreement (SLA) tra il fornitore e gli stakeholder del sistema software (ad es. clienti, operatori). In alcuni casi, i test possono continuare ad essere eseguiti dopo che il software è stato rilasciato in produzione, spesso da un team o da un'organizzazione separata. Questo accade normalmente per il performance testing e il testing di affidabilità, che possono mostrare risultati differenti nell'ambiente di produzione rispetto a quelli ottenuti nell'ambiente di test.

4.2 Aspetti Generali di Pianificazione

La mancata pianificazione per il testing non funzionale può mettere fortemente a rischio il successo di un progetto. Il Test Manager può quindi chiedere al Technical Test Analyst di identificare i principali rischi per le caratteristiche di qualità più rilevanti (si veda la tabella nel paragrafo 4.1) e di indirizzare eventuali problemi di pianificazione associati ai test proposti. Queste informazioni possono essere utilizzate per creare il master test plan.

Nell'eseguire queste attività vengono considerati i seguenti fattori generali:

- Requisiti degli stakeholder
- Requisiti dell'ambiente di test
- Necessità di strumenti e di formazione
- Considerazioni organizzative
- Considerazioni sulla sicurezza e protezione dei dati

4.2.1 Requisiti degli Stakeholder

I requisiti non funzionali sono spesso poco definiti nelle specifiche o addirittura inesistenti. Nella fase di pianificazione, i Technical Test Analyst devono essere in grado di ottenere dagli stakeholder livelli di aspettative relativi alle caratteristiche di qualità tecniche, e valutare i rischi che questi rappresentano.

Un approccio comune è di assumere che, se il cliente è soddisfatto della versione esistente del sistema, continuerà a essere soddisfatto delle nuove versioni, a condizione che vengano mantenuti i livelli di qualità raggiunti. Questo consente di utilizzare la versione esistente del sistema come benchmark. Questo può essere un approccio particolarmente utile da adottare per alcune delle caratteristiche di qualità non funzionali quali l'efficienza delle prestazioni, in cui gli stakeholder possono avere difficoltà a specificare i loro requisiti.

Quando si identificano i requisiti non funzionali, è consigliabile raccogliere diversi punti di vista da parte degli stakeholder, ad es. clienti, product owner, utenti, staff operativo e staff di manutenzione. Se vengono esclusi stakeholder chiave, alcuni requisiti risulteranno probabilmente mancanti. Per maggiori dettagli sull'elicitazione dei requisiti, fare riferimento al Syllabus Advanced Test Manager [ISTQB_ALTM_SYL].

Nello sviluppo software Agile i requisiti non funzionali possono essere descritti come user story o essere aggiunti come vincoli non funzionali a una funzionalità specificata in use case.

4.2.2 Requisiti dell'Ambiente di Test

Molti test tecnici (ad es. test di sicurezza, test di efficienza delle prestazioni, test di affidabilità), richiedono un ambiente di test simile alla produzione per poter fornire misure realistiche. In base alle dimensioni e alla complessità del sistema sotto test, questo può avere un impatto significativo sulla pianificazione e sul budget dedicato ai test. Poiché il costo di tali ambienti può essere elevato, possono essere considerate le seguenti opzioni:

- Utilizzo dell'ambiente di produzione
- Utilizzo di una versione ridotta del sistema, tenendo conto che i risultati dei test siano sufficientemente rappresentativi del sistema di produzione
- Utilizzo di risorse basate sul cloud come alternativa all'acquisizione diretta delle risorse
- Utilizzo di ambienti virtualizzati.

I tempi di esecuzione dei test devono essere pianificati accuratamente, ed è molto probabile che alcuni test possano essere eseguiti solo in momenti specifici (ad es. in momenti di ridotto utilizzo del sistema).

4.2.3 Necessità di Acquisizione di Strumenti e Formazione

Gli strumenti sono parte dell'ambiente di test. Simulatori o strumenti commerciali sono particolarmente importanti per i test di efficienza delle prestazioni (performance test) e per particolari test di sicurezza. I Technical Test Analyst dovrebbero stimare i costi e i tempi necessari per acquisire, apprendere e implementare gli strumenti. Laddove debbano essere utilizzati strumenti specializzati, la pianificazione dovrebbe tenere conto delle curve di apprendimento per i nuovi strumenti e del costo per l'assunzione di specialisti esterni degli strumenti.

Lo sviluppo di un simulatore complesso può rappresentare un progetto di sviluppo a sé stante e dovrebbe essere pianificato in quanto tale. In particolare, il testing e la documentazione dello strumento sviluppato deve essere preso in considerazione per la schedulazione e la pianificazione delle risorse. Dovrebbero essere pianificati budget e tempi sufficienti per aggiornare e ripetere il testing del simulatore a seguito di modifiche del prodotto simulato. La pianificazione dei simulatori da utilizzare in applicazioni safety deve tenere conto del testing di accettazione e dell'eventuale certificazione del simulatore da parte di un ente indipendente.

4.2.4 Considerazioni Organizzative

I test tecnici possono comportare la misurazione del comportamento dei diversi componenti di un sistema completo (ad es. server, database, reti). Se queste componenti sono distribuite su siti e organizzazioni differenti, l'effort richiesto per pianificare e coordinare i test può essere significativo. Ad esempio, alcuni componenti software possono essere disponibili per il testing di sistema solo in particolari momenti della giornata, oppure le organizzazioni possono offrire supporto al testing solo per un numero limitato di giorni. La mancata conferma che le componenti e il personale di altre organizzazioni siano disponibili "su chiamata" per il testing (ad es. esperti "presi in prestito") può causare interruzioni critiche dei test schedulati.

4.2.5 Considerazioni sulla Sicurezza dei Dati e sulla Protezione dei Dati

In fase di pianificazione dei test dovrebbero essere considerate specifiche misure di sicurezza implementate per un sistema, per garantire che tutte le attività di test siano possibili. Ad esempio, l'uso della crittografia dei dati può rendere difficile la creazione dei dati di test e la verifica dei risultati.

Le politiche e le normative sulla protezione dei dati possono precludere la generazione di qualsiasi dato di test richiesto basato sui dati di produzione (ad es. dati personali, dati della carta di credito). Rendere anonimi i dati di test è un'attività non banale, che deve essere pianificata come parte dell'implementazione dei test.

4.3 Testing di Sicurezza

4.3.1 Motivi per Considerare il Testing di Sicurezza

Il testing di sicurezza valuta la vulnerabilità di un sistema alle minacce, tentando di compromettere la politica di sicurezza del sistema. Di seguito è riportata una lista di potenziali minacce che dovrebbero essere esaminate durante il testing di sicurezza:

- Copia non autorizzata di applicazioni o dati.
- Controllo di accessi non autorizzati (ad es. capacità di eseguire attività per le quali l'utente non dispone dei diritti). Diritti, accessi e privilegi degli utenti sono il focus di questo test. L'informazione dovrebbe essere disponibile nelle specifiche di sistema.
- Software che manifesta effetti collaterali (side-effect) indesiderati durante l'esecuzione della funzione prevista. Ad esempio, un lettore multimediale che riproduce correttamente l'audio ma scrive i file in una memoria temporanea non crittografata manifesta un side-effect che può essere sfruttato da pirati del software.
- Codice inserito in una pagina web che può essere esercitato da utenti successivi (cross-site scripting o XSS). Questo codice può essere dannoso (codice "malicious").
- Buffer overflow (buffer overrun) che può essere causato dall'inserimento, in un campo di input dell'interfaccia utente, di stringhe di lunghezza maggiore rispetto alla lunghezza che il codice può gestire correttamente. Una vulnerabilità di buffer overflow rappresenta un'opportunità per eseguire istruzioni di codice dannoso.
- Denial of service, che impedisce agli utenti di interagire con un'applicazione (ad es. sovraccaricando un server web con richieste "di disturbo").
- Intercettazione, imitazione e/o alterazione e successiva trasmissione di comunicazioni (ad es. transazioni con carta di credito) da parte di terze parti in modo tale che un utente non venga a conoscenza della presenza di quelle terze parti (attacco "man in the middle").
- Violazione dei codici di crittografia utilizzati per proteggere i dati sensibili.
- Bombe logiche (o "Easter Egg"), che possono essere inserite intenzionalmente nel codice e che si attivano solo in determinate condizioni (ad es. in una data specifica). Quando le bombe logiche si attivano, possono eseguire azioni dannose come la cancellazione di file o la formattazione dei dischi.

4.3.2 Pianificazione del Testing di Sicurezza

In generale, i seguenti aspetti sono di particolare rilevanza per la pianificazione dei test di sicurezza:

- Poiché è possibile introdurre problemi di sicurezza durante l'architettura, la progettazione e l'implementazione del sistema, il testing di sicurezza può essere schedulato per i livelli di test di componente, test di integrazione e test di sistema. A causa della natura mutevole delle minacce alla sicurezza, i test di sicurezza possono anche essere schedulati regolarmente dopo che il sistema è stato rilasciato in produzione. Questo è particolarmente vero per architetture open dinamiche come Internet of Things (IoT), dove la fase di produzione è caratterizzata da molti aggiornamenti agli elementi software e hardware utilizzati.
- Gli approcci di test proposti dal Technical Test Analyst possono includere review architetturali, review della progettazione e code review, nonché l'analisi statica del codice con strumenti di sicurezza. Questi possono essere efficaci per rilevare problemi di sicurezza che vengono facilmente ignorati durante il testing dinamico.
- Il Technical Test Analyst può essere chiamato a progettare ed eseguire determinati "attacchi" alla sicurezza (si veda di seguito) che richiedono un'accurata pianificazione e coordinamento con gli stakeholder (inclusi gli specialisti del testing di sicurezza). Altri test di sicurezza possono essere eseguiti in collaborazione con gli sviluppatori o con i Test Analyst (ad es. testing dei diritti, accessi e privilegi degli utenti).

- Un aspetto essenziale della pianificazione dei test di sicurezza è ottenere l'approvazione. Per il Technical Test Analyst, questo significa assicurare che sia stato ottenuto dal Test Manager il permesso esplicito di eseguire i test di sicurezza pianificati. Eventuali test aggiuntivi non pianificati potrebbero sembrare attacchi effettivi e la persona che conduce tali test potrebbe essere a rischio di azioni legali. Senza nulla di scritto che specifichi le intenzioni e l'autorizzazione, la scusa "Stavamo eseguendo un test di sicurezza" potrebbe essere difficile da spiegare in modo convincente.
- La pianificazione dei test di sicurezza dovrebbe essere coordinata con l'Information Security Officer di un'organizzazione, se l'organizzazione ha un tale ruolo.
- Si dovrebbe notare che i miglioramenti che possono essere apportati alla sicurezza di un sistema possono influenzarne l'efficienza delle prestazioni o l'affidabilità. Dopo aver applicato miglioramenti alla sicurezza, è consigliabile prendere in considerazione la necessità di condurre test di efficienza delle prestazioni o di affidabilità (si vedano i paragrafi 4.4 e 4.5).

Quando si esegue la pianificazione dei test di sicurezza possono essere applicati specifici standard come lo standard [IEC 62443-3-2], che si applica all'automazione industriale e ai sistemi di controllo.

Il Syllabus Security Testing [CTSL_SEC_SYL] include ulteriori dettagli sugli aspetti chiave della pianificazione dei test di sicurezza.

4.3.3 Specifica dei Test di Sicurezza

Particolari test di sicurezza possono essere raggruppati [Whittaker04] in base all'origine del rischio di sicurezza. Questi includono:

- Test relativi all'interfaccia utente: accesso non autorizzato e input dannosi.
- Test relativi al file system: accesso ai dati sensibili memorizzati in file o repository.
- Test relativi al sistema operativo: memorizzazione di informazioni sensibili come password in un formato non crittografato, che potrebbero essere esposte quando il sistema genera un crash attraverso input dannosi.
- Test relativi al software esterno: interazioni che possono verificarsi tra componenti esterne utilizzate dal sistema. Queste possono essere a livello di rete (ad es. pacchetti o messaggi errati che vengono trasmessi) o a livello di componente software (ad es. failure di un componente software su cui si basa il software).

Anche le sotto-caratteristiche ISO 25010 [ISO25010] di sicurezza possono fornire una base per definire le specifiche dei test di sicurezza. Queste si focalizzano su specifici aspetti di sicurezza:

- Confidenzialità
- Integrità
- Non-repudiation
- Responsabilità
- Autenticità

Per lo sviluppo dei test di sicurezza è possibile utilizzare il seguente approccio [Whittaker04]:

- Raccogliere informazioni che possono essere utili per specificare i test, come nomi di dipendenti, indirizzi fisici, dettagli relativi a reti interne, numeri IP, identità del software o dell'hardware utilizzato, e versione del sistema operativo.
- Eseguire una scansione delle vulnerabilità utilizzando gli strumenti ampiamente disponibili. Tali strumenti non vengono utilizzati direttamente per compromettere il sistema (o sistemi), ma per identificare le vulnerabilità che sono o che possono risultare in una violazione della politica di sicurezza. Vulnerabilità specifiche possono anche essere identificate utilizzando informazioni e checklist, come quelle fornite da National Institute of Standards and Technology (NIST) [Web-1] e da Open Web Application Security Project™ (OWASP) [Web-4].
- Sviluppare "piani di attacco" (cioè un piano di azioni di test intese a compromettere la politica di sicurezza di un particolare sistema) utilizzando le informazioni raccolte. Per rilevare i difetti di

sicurezza più gravi, nei piani di attacco occorre definire diversi input attraverso diverse interfacce (ad es. interfaccia utente, file system). I diversi "attacchi" descritti in [Whittaker04] sono una preziosa fonte di tecniche sviluppate specificamente per il testing di sicurezza.

Si noti che è possibile sviluppare piani di attacco per il penetration testing.

Il paragrafo 3.2 (analisi statica) e il Syllabus Security Testing [CTSL_SEC_SYL] includono ulteriori dettagli del testing di sicurezza.

4.4 Testing di Affidabilità

4.4.1 Introduzione

La classificazione ISO 25010 delle caratteristiche di qualità del prodotto definisce le seguenti sotto-caratteristiche di affidabilità: maturità, tolleranza ai guasti, recuperabilità e disponibilità. Il testing di affidabilità riguarda la capacità di un sistema o di un software di eseguire specifiche funzioni in determinate condizioni per un determinato periodo di tempo.

4.4.2 Testing di Maturità

La maturità è il grado in cui il sistema (o software) soddisfa i requisiti di affidabilità in condizioni operative normali, che sono normalmente specificate utilizzando un profilo operativo (si veda il paragrafo 4.9). Le misure di maturità, quando utilizzate, spesso costituiscono uno dei criteri di rilascio per un sistema.

Tradizionalmente, la maturità viene specificata e misurata per sistemi ad alta affidabilità, come quelli associati a funzioni safety-critical (ad es. un sistema di controllo dei voli aerei), dove l'obiettivo di maturità viene definito come parte di uno standard normativo. Un requisito di maturità per tale sistema ad alta affidabilità può essere un Mean Time Between Failures (MTBF) fino a 10^9 ore (sebbene questo sia praticamente impossibile da misurare).

L'approccio usuale al testing di maturità per i sistemi ad alta affidabilità è noto come modello di crescita dell'affidabilità (reliability growth model) e normalmente avviene al termine del testing di sistema, dopo che il testing di altre caratteristiche di qualità è stato completato e sono stati corretti eventuali difetti associati ai failure rilevati . È un approccio statistico solitamente eseguito in un ambiente di test il più vicino possibile all'ambiente operativo. Per misurare un MTBF specifico, vengono generati input di test in base al profilo operativo, il sistema viene eseguito e i failure vengono registrati (e successivamente corretti). Una frequenza di failure ridotta consente di prevedere MTBF utilizzando un modello di crescita dell'affidabilità.

Quando la maturità viene utilizzata come obiettivo per sistemi a minore affidabilità (ad es. sistemi non safety), il numero di failure osservati durante un periodo definito di utilizzo operativo previsto (ad es. non più di 2 failure ad alto impatto a settimana) può essere utilizzato e può essere registrato come parte del service level agreement (SLA) per il sistema.

4.4.3 Testing di Disponibilità

La disponibilità è in genere specificata in termini di quantità di tempo in cui un sistema (o software) è disponibile per gli utenti e per altri sistemi in condizioni operative normali. I sistemi possono avere una maturità bassa, ma hanno comunque un'elevata disponibilità. Ad esempio, una rete telefonica potrebbe fallire nel connettere diverse chiamate (e quindi avere una maturità bassa), ma finché il sistema si ripristina rapidamente e consente successivi tentativi di connessione, la maggior parte degli utenti sarà soddisfatta. Tuttavia, un singolo failure che causasse un'interruzione della rete telefonica per diverse ore rappresenterebbe un livello di disponibilità non accettabile. La disponibilità viene spesso specificata come parte di uno SLA e misurata per i sistemi che sono operativi, come siti Web e applicazioni SaaS

(Software as a Service). La disponibilità di un sistema può essere descritta come 99,999% ("five nines"), e in questo caso dovrebbe essere non disponibile per non più di 5 minuti all'anno; in alternativa. la disponibilità del sistema può essere specificata in termini di indisponibilità (ad es. il sistema non dovrà essere inutilizzabile per più di 60 minuti al mese).

La misurazione della disponibilità prima che il sistema sia operativo (ad es. come parte della decisione di un rilascio) viene spesso eseguita utilizzando gli stessi test eseguiti per misurare la maturità; i test si basano su un profilo operativo di utilizzo atteso per un periodo prolungato, e vengono eseguiti in un ambiente di test il più vicino possibile all'ambiente operativo. La disponibilità può essere misurata come MTTF/(MTTF + MTTR), dove MTTF è il Mean Time to Failure e MTTR è il Mean Time to Repair, che viene spesso misurato come parte dei test di manutenibilità. In un sistema ad alta affidabilità che incorpora la recuperabilità (si veda il paragrafo 4.4.5), quando il sistema impiega del tempo per riprendersi da un failure è possibile sostituire nell'equazione il parametro MTTR con Mean Time To Recover.

4.4.4 Testing di Tolleranza ai Guasti

I sistemi (o software) con requisiti di affidabilità estremamente elevati spesso incorporano una progettazione di tolleranza ai guasti, consentendo idealmente al sistema di continuare a funzionare senza tempi di fermo evidenti in caso di failure. La misura principale della tolleranza ai guasti per un sistema è la capacità del sistema di tollerare i failure. Quindi, il testing di tolleranza ai guasti implica la simulazione dei failure per determinare se il sistema può continuare a funzionare quando si verifica tale failure. L'identificazione di potenziali condizioni di failure da testare è una parte importante del testing di tolleranza ai guasti.

Una progettazione della tolleranza ai guasti in genere coinvolge uno o più sottosistemi duplicati, fornendo così un livello di ridondanza in caso di failure. Nel caso del software, tali sistemi duplicati necessitano di essere sviluppati in modo indipendente, per evitare failure; questo approccio è noto come programmazione N-version. I sistemi di controllo dei voli aerei possono includere tre o quattro livelli di ridondanza, con le funzioni più critiche implementate in diverse varianti. Laddove l'affidabilità hardware è un problema, un sistema embedded può essere eseguito su più processori diversi, mentre un sito Web critico può essere eseguito con un server mirror (failover) che esegue le stesse funzioni ed è sempre disponibile in caso di guasto del server primario. Qualunque sia l'approccio alla tolleranza agli errori implementato, il testing richiede in genere sia il rilevamento del failure sia la successiva risposta al failure da testare.

Il testing di fault injection verifica la robustezza di un sistema in presenza di difetti nell'ambiente del sistema (ad es. un alimentatore difettoso, messaggi di input non formattati correttamente, un processo o un servizio non disponibile, un file non trovato oppure memoria non disponibile) e difetti nel sistema stesso (ad es. un flipped bit causato da radiazioni cosmiche, una progettazione scadente o una codifica errata). Il testing del faut injection è un tipo di test negativo: vengono iniettati deliberatamente i difetti nel sistema per assicurare che il sistema reagisca nel modo previsto (cioè in modo safely per un sistema safety). A volte gli scenari di difettosità che vengono testati non dovrebbero mai verificarsi (ad es. un'attività software non dovrebbe mai "morire" o rimanere bloccata in un loop infinito) e non possono essere simulati dal testing di sistema tradizionale, ma con il testing di fault injection viene creato uno scenario del difetto e viene misurato il successivo comportamento del sistema per assicurarsi che rilevi e gestisca il failure.

4.4.5 Testing di Recuperabilità

La recuperabilità è una misura della capacità di un sistema (o di un software) di recuperare (recovery) da un failure, sia in termini di tempo richiesto per il ripristino (che può essere uno stato operativo ridotto) sia in termini di quantità di dati persi. Gli approcci al testing di recuperabilità includono il testing di failover e il testing di backup and restore; entrambi includono tipicamente procedure di test basate su dry run e in modo occasionale, e ideale, test pratici senza preavviso in ambienti operativi.

Il testing di backup and restore si focalizza sul testing delle procedure in atto per minimizzare gli effetti di un failure sui dati di sistema. I test valutano le procedure sia per il backup che per il restore (ripristino) dei dati. Sebbene il testing per il backup dei dati sia relativamente semplice, il test per il restore di un sistema dai dati di backup può essere più complesso e spesso richiede un'accurata pianificazione, per garantire che l'interruzione dell'operatività del sistema sia ridotta al minimo. Le misure includono il tempo impiegato per eseguire diversi tipi di backup (ad es. completo e incrementale), il tempo impiegato per ripristinare i dati (obiettivo del tempo di recovery) e il livello di perdita dei dati ritenuto accettabile (obiettivo del punto di recovery).

Il testing di failover viene eseguito quando l'architettura di sistema comprende un sistema primario e un sistema failover che subentrerà in caso di failure del sistema primario. Laddove un sistema deve essere in grado di recuperare da un failure catastrofico (ad es. un'inondazione, un attacco terroristico o un grave attacco ransomware), il testing di failover è spesso noto come testing di disaster recovery e il sistema (o i sistemi) failover può spesso trovarsi in un altro sito geografico. L'esecuzione di un test completo di disaster recovery su un sistema che è operativo richiede una pianificazione estremamente accurata, a causa dei rischi e delle interruzioni (spesso correlata al tempo libero dei senior manager, che è probabile venga considerato nella gestione del recovery). Se un test completo di disaster recovery fallisce, si tornerà immediatamente al sistema primario (poichè non è stato realmente distrutto!). I test di failover includono il testing di passaggio al sistema failover che, una volta diventato operativo, fornisce il service level (livello di servizio) richiesto.

4.4.6 Pianificare il Testing di Affidabilità

In generale, i seguenti elementi sono di particolare rilevanza nella pianificazione dei test di affidabilità:

- Tempistica - Il testing di affidabilità in genere richiede che il sistema da testare sia completo e che gli altri tipi di test siano già stati completati, e l'esecuzione può richiedere molto tempo.
- Costi - I sistemi ad alta affidabilità sono notoriamente costosi da testare a causa dei lunghi periodi per i quali i sistemi devono essere testati senza failure, per poter prevedere un MTBF elevato richiesto.
- Durata - Il testing di maturità che utilizza modelli di crescita dell'affidabilità si basa sui failure rilevati e, per livelli di affidabilità elevati, occorrerà molto tempo per ottenere risultati statisticamente significativi.
- Ambiente di test - L'ambiente di test richiede di essere il più similare possibile a quello operativo, oppure può essere utilizzato l'ambiente operativo. Tuttavia, se si utilizza l'ambiente operativo, questo può essere dannoso per gli utenti e può essere ad alto rischio se, ad esempio, un test di disaster recovery influisce negativamente sul sistema che è operativo.
- Ambito - Diversi sottosistemi e componenti possono essere testati per differenti tipi e livelli di affidabilità.
- Criteri di uscita - I requisiti di affidabilità dovrebbero essere stabiliti da standard normativi per le applicazioni safety.
- Failure - Le misure di affidabilità dipendono molto dal numero di failure e quindi deve esistere un accordo iniziale su cosa è definito come failure.
- Sviluppatori - Il testing di maturità che utilizza modelli di crescita dell'affidabilità richiede di raggiungere un accordo con gli sviluppatori affinché i difetti identificati vengano corretti il prima possibile.
- La misurazione dell'affidabilità operativa è relativamente semplice rispetto alla misurazione dell'affidabilità prima del rilascio, poiché devono essere misurati solo i failure; questo può richiedere una collaborazione con lo staff di Operations.
- Testing anticipato - Raggiungere un'elevata affidabilità (anziché misurare l'affidabilità) richiede di iniziare il testing il prima possibile, con review rigorose dei primi documenti della baseline e con l'analisi statica del codice.

4.4.7 Specifica dei Test di Affidabilità

Per il testing di maturità e disponibilità, il testing si basa in gran parte sul testing del sistema in condizioni operative normali. Per tali test è richiesto un profilo operativo che definisca come ci si aspetta debba essere utilizzato il sistema. Si veda il paragrafo 4.9 per maggiori dettagli sui profili operativi.

Per il testing di tolleranza ai guasti e per il testing di recuperabilità, è spesso necessario generare test che replichino i failure nell'ambiente e nel sistema stesso, per determinare come reagisce il sistema. Il testing di fault injection viene spesso utilizzato per questo. Sono disponibili diverse tecniche e checklist per l'identificazione di possibili difetti e failure corrispondenti (ad es. Fault Tree Analysis, Failure Mode and Effect Analysis).

4.5 Performance Testing (Testing di Efficienza delle Prestazioni)

4.5.1 Introduzione

La classificazione ISO 25010 delle caratteristiche di qualità del prodotto definisce le seguenti sotto-caratteristiche dell'efficienza delle prestazioni: comportamento nel tempo, utilizzo delle risorse, e capacità. Il performance testing, associato alla caratteristica di qualità di efficienza delle prestazioni, riguarda la misurazione delle prestazioni di un sistema o di un software sotto specifiche condizioni rispetto alla quantità di risorse utilizzate. Le risorse tipiche includono l'elapsed time, il tempo di CPU, la memoria e la larghezza di banda.

4.5.2 Testing del Comportamento nel Tempo

Il testing del comportamento nel tempo misura i seguenti aspetti di un sistema (o software) sotto specifiche condizioni operative:

- Elapsed time: tempo trascorso dalla ricezione di una richiesta fino alla prima risposta (cioè il tempo per iniziare a rispondere, non il tempo per completare l'attività richiesta), chiamato anche tempo di risposta.
- Tempo di turnaround dall'inizio di un'attività fino al completamento dell'attività, detto anche tempo di elaborazione.
- Numero di attività completate per unità di tempo (ad es. numero di operazioni del database al secondo), chiamato anche throughput rate.

Per molti sistemi, i tempi di risposta massimi per le diverse funzioni di sistema sono specificati come requisiti. In tali casi il tempo di risposta è la somma dell'elapsed time e del tempo di turnaround. Quando un sistema deve eseguire un numero di passi (ad es. una pipeline) per completare un'attività, può essere utile misurare il tempo impiegato per ogni passo e analizzare i risultati per determinare se uno o più passi stanno causando un collo di bottiglia.

4.5.3 Testing di Utilizzo delle Risorse

Il testing di utilizzo delle risorse misura i seguenti aspetti di un sistema (o software) in condizioni operative specificate:

- Utilizzo della CPU, normalmente inteso come percentuale del tempo di CPU disponibile.
- Utilizzo della memoria, normalmente inteso come percentuale della memoria disponibile.
- Utilizzo del dispositivo di I/O, normalmente inteso come percentuale del tempo disponibile del dispositivo di I/O.
- Utilizzo della larghezza di banda, normalmente inteso come percentuale della larghezza di banda disponibile.

4.5.4 Testing della Capacità

Il testing della capacità misura i limiti massimi per i seguenti aspetti di un sistema (o software) in condizioni operative specificate:

- Transazioni processate per unità di tempo (ad es. un massimo di 687 parole tradotte al minuto).
- Utenti che accedono simultaneamente al sistema (ad es. un massimo di 1223 utenti).
- Nuovi utenti aggiunti, per avere accesso al sistema, per unità di tempo (ad es. un massimo di 400 utenti aggiunti al secondo).

4.5.5 Aspetti Comuni del Performance Testing

Quando si esegue il testing del comportamento nel tempo, dell'utilizzo delle risorse o della capacità, è normale che vengano eseguite diverse misurazioni e che la media venga utilizzata come misura per il reporting; questo perché i valori temporali misurati possono variare in base alle altre attività in background che il sistema potrebbe eseguire. In alcune situazioni, le misurazioni verranno gestite in modo più meticoloso (ad es. utilizzando la varianza o altre misure statistiche), oppure i valori anomali possono essere esaminati e scartati, se ritenuto appropriato.

L'analisi dinamica (si veda il paragrafo 3.3.4) può essere utilizzata per identificare i componenti che causano un collo di bottiglia, per misurare le risorse utilizzate per il testing di utilizzo delle risorse e per misurare i limiti massimi per il testing della capacità.

4.5.6 Tipi di Performance Test

Il performance testing differisce dalla maggior parte degli altri tipi di test, in quanto possono esistere due obiettivi distinti. Il primo è quello di determinare se il software sotto test soddisfa i criteri di accettazione specificati. Ad es. determinare se il sistema visualizza una pagina Web richiesta entro il tempo massimo specificato di 4 secondi. Il secondo obiettivo è quello di fornire informazioni agli sviluppatori del sistema per aiutarli a migliorare l'efficienza del sistema. Ad esempio, rilevare i colli di bottiglia e identificare quali parti dell'architettura di sistema sono influenzate negativamente quando un numero inaspettatamente elevato di utenti accede al sistema contemporaneamente.

Il performance testing descritto nei paragrafi 4.5.2, 4.5.3 e 4.5.4 può essere utilizzato per determinare se il software sotto test soddisfa i criteri di accettazione specificati. Vengono anche utilizzati per misurare i valori della baseline che saranno considerati per un confronto successivo, quando il sistema viene modificato. I seguenti tipi di performance test sono i più utilizzati per fornire informazioni agli sviluppatori su come il sistema risponde sotto specifiche e differenti condizioni operative.
FINO QUI

Testing di Carico

Il testing di carico si focalizza sulla capacità di un sistema di gestire carichi differenti. Questi carichi sono generalmente definiti in termini di numero di utenti che accedono al sistema simultaneamente o in termini di numero di processi concorrenti in esecuzione, e possono essere definiti come profili operativi (si veda il paragrafo 4.9 per maggiori dettagli sui profili operativi). La gestione di questi carichi viene tipicamente misurata in termini di comportamento nel tempo e di utilizzo delle risorse del sistema (ad es. determinare l'effetto sui tempi di risposta raddoppiando il numero di utenti). Quando si esegue il testing di carico, è normale pratica iniziare con un carico basso e aumentare gradualmente il carico misurando il comportamento del sistema nel tempo e l'utilizzo delle risorse. Le informazioni tipiche ottenute dal testing di carico che potrebbero essere utili agli sviluppatori includono modifiche impreviste nei tempi di risposta o l'utilizzo delle risorse di sistema quando il sistema ha gestito un carico particolare.

Stress Testing

Esistono due tipi di stress test; il primo tipo è simile al testing di carico e il secondo tipo è una forma di testing di robustezza.

Nel primo tipo, i test di carico vengono normalmente eseguiti con il carico inizialmente impostato al massimo previsto e poi aumentato fino a quando il sistema fallisce (ad es. i tempi di risposta diventano irragionevolmente lunghi o il sistema genera un crash). A volte, invece di forzare il sistema a fallire, viene utilizzato un carico elevato per sollecitare il sistema, quindi il carico viene ridotto a un livello normale e il sistema viene verificato per assicurare che i suoi livelli di prestazione siano tornati ai livelli precedenti alla sollecitazione.

Nel secondo tipo, i performance test vengono eseguiti con il sistema deliberatamente compromesso, riducendo il suo accesso alle risorse previste (ad es. riducendo la memoria disponibile o la larghezza di banda). I risultati dello stress testing possono fornire agli sviluppatori un'idea di quali aspetti del sistema sono più critici (ad es. link deboli) e quindi può essere necessario un upgrade.

Testing di Scalabilità

Un sistema scalabile può adattarsi a carichi differenti. Ad esempio, un sito web scalabile potrebbe essere scalato per utilizzare più server di back-end all'aumentare della domanda, e utilizzarne di meno quando la domanda diminuisce. Il testing di scalabilità è simile al testing di carico, ma verifica la capacità di un sistema di aumentare e diminuire la scalabilità in base a carichi variabili (ad es. più utenti di quanto l'attuale hardware possa gestire).

Il Syllabus Performance Testing [CTSL_PT_SYL] include ulteriori tipi di perfomance test.

4.5.7 Pianificazione dei Performance Test

In generale, i seguenti elementi sono di particolare rilevanza quando si pianificano i performance test:

- Tempistica - I performance test spesso richiedono che il sistema completo sia implementato ed eseguito in un ambiente di test rappresentativo, il che significa che in genere vengono eseguiti come parte del testing di sistema.
- Review - Le code review, in particolare quelle che si focalizzano sull'interazione del database, sull'interazione dei componenti e sulla gestione degli errori, possono identificare problemi di efficienza delle prestazioni (in particolare relativi alla logica di "wait and retry" e alle query inefficienti) e dovrebbero essere schedulate per essere eseguite non appena il codice è disponibile (cioè prima del testing dinamico).
- Testing anticipato - Alcuni performance test (ad es. che determinano l'uso di CPU per un componente critico) possono essere schedulati come parte del testing di componente. I componenti, identificati dai performance test come colli di bottiglia, possono essere aggiornati e testati di nuovo separatamente come parte del testing di componente.
- Modifiche architetturali - Risultati negativi del performance testing possono talvolta comportare una modifica dell'architettura di sistema. Dove queste importanti modifiche al sistema potrebbero essere suggerite dai risultati dei performance test, il performance testing dovrebbe iniziare il prima possibile, per massimizzare il tempo disponibile ad indirizzare tali problemi.
- Costi - Gli strumenti e gli ambienti di test possono essere costosi, il che significa che è possibile affittare ambienti di test cloud-based temporanei e utilizzare licenze per strumenti "top-up". In tali casi, la pianificazione dei test in genere richiede di ottimizzare il tempo impiegato per l'esecuzione dei test, in modo da minimizzare i costi.
- Ambiente di test - L'ambiente di test richiede di essere il più rappresentativo possibile dell'ambiente operativo, altrimenti aumenta la sfida di riuscire a scalare i risultati dei performance test dall'ambiente di test all'ambiente operativo previsto.
- Criteri di uscita - A volte, i requisiti di efficienza delle prestazioni possono essere difficili da ottenere dal cliente e quindi sono spesso derivati da baseline di sistemi precedenti o similari.

Nel caso di sistemi safety embedded, alcuni requisiti, come la quantità massima di CPU e la memoria utilizzata, possono essere specificati dagli standard normativi.

- Strumenti - Sono spesso necessari strumenti di generazione del carico per supportare il performance testing. Ad esempio, la verifica della scalabilità di un sito web popolare può richiedere la simulazione di centinaia di migliaia di utenti virtuali. Gli strumenti che simulano le restrizioni sulle risorse sono anche particolarmente utili per lo stress testing. Dovrebbe essere garantito che tutti gli strumenti acquisiti per supportare il testing siano compatibili con i protocolli di comunicazione utilizzati dal sistema sotto test.

Il Syllabus Performance Testing [CTSL_PT_SYL] include ulteriori dettagli sulla pianificazione dei performance test.

4.5.8 Specifica dei Performance Test

Il performance testing si basa in gran parte sul testing del sistema sotto specifiche condizioni operative. Per tali test, è richiesto un profilo operativo che definisca come si prevede di utilizzare il sistema. Si veda il paragrafo 4.9 per maggiori dettagli sui profili operativi.

Per il performance testing, è spesso necessario cambiare il carico sul sistema modificando parti del profilo operativo, in modo da simulare un cambiamento rispetto all'uso operativo atteso del sistema. Ad esempio, nel caso del testing della capacità, sarà necessario modificare il profilo operativo aumentando il valore della variabile relativa alla capacità sotto test (ad es. aumentando il numero di utenti che accedono al sistema fino a quando il sistema non smette di rispondere, per determinare la capacità di accesso degli utenti). Allo stesso modo, il volume delle transazioni può essere aumentato gradualmente durante il testing di carico.

Il Syllabus Performance Testing [CTSL_PT_SYL] include ulteriori dettagli sulla progettazione dei test di efficienza delle prestazioni.

4.6 Testing di Manutenibilità

Durante il ciclo di vita, il software spesso richiede sostanzialmente un tempo di manutenzione superiore al tempo di sviluppo. Per garantire che l'attività di manutenzione sia il più efficiente possibile, viene eseguito il testing di manutenibilità per misurare la facilità con cui il codice può essere analizzato, modificato, testato, modularizzato e riutilizzato. Il testing di manutenibilità non dovrebbe essere confuso con il testing di manutenzione, che viene eseguiti per testare le modifiche apportate al software operativo.

Gli obiettivi tipici di manutenibilità per gli stakeholder impattati (ad es..il proprietario del software o l'operatore) includono:

- Minimizzare il costo di ownership o di funzionamento del software.
- Minimizzare il downtime richiesto per la manutenzione software.

I test di manutenibilità dovrebbero essere inclusi in un approccio di test in cui si applicano uno o più dei seguenti fattori:

- Sono probabili modifiche software dopo il rilascio del software in produzione (ad es. per correggere difetti o introdurre aggiornamenti pianificati).
- I benefici di raggiungere gli obiettivi di manutenibilità durante il ciclo di vita dello sviluppo software vengono considerati, dagli stakeholder impattati, superiori ai costi di esecuzione dei test di manutenibilità e di implementazione delle modifiche richieste.
- I rischi di una scarsa manutenibilità del software (ad es. lunghi tempi di risposta per i difetti segnalati da utenti e/o clienti) giustificano l'esecuzione dei test di manutenibilità.

4.6.1 Testing di Manutenibilità Statico e Dinamico

Le tecniche appropriate per il testing statico di manutenibilità includono l'analisi statica e le review, come discusso nei paragrafi 3.2 e 5.2. Il testing di manutenibilità dovrebbe essere avviato non appena la documentazione di progetto è disponibile, e dovrebbe continuare durante l'implementazione del codice. Poiché la manutenibilità è incorporata nel codice e nella documentazione di ogni componente del codice, la manutenibilità può essere valutata nelle fasi iniziali del ciclo di vita dello sviluppo software senza dover attendere un sistema completo e funzionante.

Il testing dinamico di manutenibilità si focalizza sulle procedure documentate che sono state sviluppate per manutenere una particolare applicazione (ad es. per eseguire aggiornamenti software). Gli scenari di manutenzione vengono utilizzati come test case, per garantire che i service level richiesti siano raggiungibili con le procedure documentate. Questa forma di test è particolarmente rilevante quando l'infrastruttura sottostante è complessa e le procedure di supporto possono coinvolgere più dipartimenti/organizzazioni. Questa forma di test può essere eseguita come parte del testing di accettazione operativo.

4.6.2 Sotto-caratteristiche di Manutenibilità

La manutenibilità di un sistema può essere misurata in termini di:

- Analizzabilità
- Modificabilità
- Testabilità

I fattori che influenzano queste caratteristiche includono l'applicazione di buone pratiche di programmazione (ad es. commenti, naming delle variabili, indentazione) e la disponibilità di documentazione tecnica (ad es. specifiche di progettazione del sistema, specifiche di interfaccia).

Altre sotto-caratteristiche di qualità rilevanti per la manutenibilità [ISO25010] sono:

- Modularità
- Riutilizzabilità

La modularità può essere testata attraverso l'analisi statica (si veda il paragrafo 3.2.3). Il testing di riutilizzabilità può assumere la forma di review architetturali (si veda il Capitolo 5).

4.7 Testing di Portabilità

4.7.1 Introduzione

In generale, il testing di portabilità è relativo al grado in cui un componente o un sistema software può essere trasferito nell'ambiente target (inizialmente o da un ambiente esistente), può essere adattato al nuovo ambiente, o può sostituire un'altra entità.

Lo standard [ISO25010] include le seguenti sotto-caratteristiche di portabilità:

- Installabilità
- Adattabilità
- Sostituibilità

Il testing di portabilità può iniziare con componenti singole (ad es. la sostituibilità di una particolare componente, come il cambiamento da un database management system ad un altro) e successivamente espandere l'ambito man mano che diventa disponibile il codice. L'installabilità può non essere testabile finché tutti i componenti del prodotto non sono funzionanti.

La portabilità deve essere progettata e integrata nel prodotto e quindi deve essere presa in considerazione nelle prime fasi dell'architettura e della progettazione. Le review dell'architettura e della progettazione possono essere particolarmente produttive per identificare potenziali requisiti e problemi di portabilità (ad es. la dipendenza da un particolare sistema operativo).

4.7.2 Testing di Installabilità

Il testing di installabilità viene eseguito sul software e sulle procedure scritte utilizzate per installare il software nell'ambiente target. Questo può includere, ad esempio, il software sviluppato per installare un sistema operativo oppure un "wizard" (procedura guidata) di installazione per installare un prodotto su un PC client.

Obiettivi tipici del testing di installabilità includono:

- Validare che il software può essere installato seguendo le istruzioni in un manuale di installazione (inclusa l'esecuzione di eventuali script di installazione) o utilizzando un wizard di installazione. Questo significa esercitare le opzioni di installazione per diverse configurazioni hardware/software e per diversi gradi di installazione (ad es. iniziale o update).
- Verificare se i failure che si verificano durante l'installazione (ad es. failure durante il caricamento di particolari DLL) vengono gestiti correttamente dal software di installazione senza lasciare il sistema in uno stato indefinito (ad es. software parzialmente installato o configurazioni di sistema errate).
- Verificare se è possibile completare un'installazione/disinstallazione parziale.
- Verificare se un wizard di installazione è in grado di identificare piattaforme hardware o configurazioni del sistema operativo non valide.
- Misurare se il processo di installazione può essere completato entro un numero specificato di minuti o entro un numero specificato di passi.
- Validare che il software può essere disinstallato o portato alla versione precedente ("downgrade").

Il testing di adeguatezza funzionale viene generalmente eseguito dopo il testing di installabilità, per rilevare eventuali difetti introdotti dall'installazione (ad es. configurazioni errate, funzioni non disponibili). Il testing di usabilità viene normalmente eseguito in parallelo al testing di installabilità (ad es. per validare che agli utenti vengano fornite istruzioni e messaggi di feedback/errore comprensibili durante l'installazione).

4.7.3 Testing di Adattabilità

Il testing di adattabilità verifica se una determinata applicazione può funzionare correttamente in tutti gli ambienti target previsti (hardware, software, middleware, sistema operativo, ecc.). La specifica dei test di adattabilità richiede che gli ambienti target previsti vengano identificati, configurati e resi disponibili al team di test. Questi ambienti vengono poi testati utilizzando una selezione di test case funzionali che esercitano le varie componenti presenti nell'ambiente.

L'adattabilità può riguardare la capacità del software di essere portato in determinati ambienti eseguendo una procedura predefinita. I test possono valutare questa procedura.

4.7.4 Testing di Sostituibilità

Il testing di sostituibilità si concentra sulla capacità di un componente software di sostituire un componente software esistente in un sistema. Questo può essere particolarmente rilevante per sistemi che utilizzano software commerciale off-the-shelf (COTS, Commercial off-the-shelf) per specifici componenti di sistema o per applicazioni IoT.

I test di sostituibilità possono essere eseguiti in parallelo ai test di integrazione funzionale, quando sono disponibili componenti alternativi per l'integrazione nel sistema completo. La sostituibilità può anche

essere valutata da review tecniche o ispezioni a livello architetturale e di progettazione, dove viene data enfasi alla chiara definizione delle interfacce del componente che può essere usato per la sostituzione.

4.8 Testing di Compatibilità

4.8.1 Introduzione

Il testing di compatibilità prende in considerazione i seguenti aspetti [ISO25010]:

- Coesistenza
- Interoperabilità

4.8.2 Testing di Coesistenza

I sistemi informatici che non sono correlati tra loro si dicono coesistenti quando possono essere eseguiti nello stesso ambiente (ad es. sullo stesso hardware) senza influenzare il comportamento dell'altro (ad es. conflitti di risorse). Il testing di coesistenza dovrebbe essere eseguito quando un software nuovo o aggiornato deve essere distribuito in ambienti che contengono già applicazioni installate.

Problemi di coesistenza possono verificarsi quando l'applicazione viene testata in un ambiente dove è l'unica applicazione installata (dove i problemi di incompatibilità non sono rilevabili) e quindi distribuita in un altro ambiente (ad es. produzione) che esegue anche altre applicazioni.

Gli obiettivi tipici del testing di coesistenza includono:

- Valutazione del possibile impatto negativo sull'adeguatezza funzionale quando le applicazioni vengono caricate nello stesso ambiente (ad es. utilizzo di risorse in conflitto quando un server esegue più applicazioni)
- Valutazione su qualsiasi applicazione dell'impatto causato dalla distribuzione di fix e aggiornamenti del sistema operativo

I problemi di coesistenza dovrebbero essere analizzati durante la pianificazione dell'ambiente target di produzione, ma i test effettivi vengono normalmente eseguiti dopo il completamento del testing di sistema.

4.9 Profili Operativi

I profili operativi vengono utilizzati come parte delle specifiche di test per diversi tipi di test non funzionali, inclusi il testing di affidabilità e il performance testing. Sono particolarmente utili quando il requisito da testare include il vincolo di "sotto specifiche condizioni" in quanto possono essere utilizzati per definire queste condizioni.

Il profilo operativo definisce un modello di utilizzo del sistema, tipicamente in termini di utenti del sistema e di operazioni eseguite dal sistema. Gli utenti sono in genere specificati in termini di quanti dovrebbero ci si aspetta utilizzino il sistema (e in quali momenti) ed eventualmente rispetto al loro tipo (ad esempio, utente principale, utente secondario). Tipicamente vengono specificate le diverse operazioni che ci si aspetta vengano eseguite dal sistema, con la loro frequenza (e probabilità di accadimento). Queste informazioni possono essere ottenute utilizzando strumenti di monitoraggio (laddove l'applicazione effettiva o similare sia già disponibile) o prevedendo l'utilizzo sulla base di algoritmi o stime fornite dall'organizzazione aziendale.

Gli strumenti possono essere utilizzati per generare input di test in base al profilo operativo, spesso utilizzando un approccio che genera gli input dei test in modo pseudo-casuale. Tali strumenti possono essere utilizzati per creare utenti "virtuali" o simulati in quantità che corrispondono al profilo operativo

(ad es. per il testing di affidabilità e disponibilità) o lo superano (ad es. per lo stress testing o il testing della capacità). Si veda il paragrafo 6.2.3 per maggiori dettagli su questi strumenti

5. Review - 165 minuti

Parole Chiave
review, review tecnica

Obiettivi di Apprendimento per le Review

5.1 I Compiti del Technical Test Analyst nelle Review

TTA-5.1.1 (K2) Spiegare perché la preparazione della review è importante per il Technical Test Analyst

5.2 Utilizzo delle Checklist nelle Review

TTA-5.2.1 (K4) Analizzare una progettazione architetturale e identificare problemi in base a una checklist fornita nel Syllabus

TTA-5.2.2 (K4) Analizzare una porzione di codice o pseudo-codice e identificare problemi in base a una checklist fornita nel Syllabus

5.1 I Compiti del Technical Test Analyst nelle Review

I Technical Test Analyst devono essere partecipanti attivi nel processo di review tecnica, fornendo il loro unico punto di vista. Tutti i partecipanti alla review dovrebbero avere una formazione sulle review formali per comprendere meglio i propri ruoli, e devono impegnarsi per ottenere i benefici di una review tecnica ben condotta. Questo include mantenere un rapporto di lavoro costruttivo con gli autori quando si descrivono e si discutono i commenti della review. Per una descrizione dettagliata delle review tecniche, incluse numerose checklist di review, vedere [Wiegers02]. I Technical Test Analyst normalmente partecipano a review tecniche e ispezioni, portando un punto di vista operativo (comportamentale) che potrebbe non essere considerato dagli sviluppatori. I Technical Test Analyst svolgono un ruolo importante nella definizione, applicazione e manutenzione delle checklist di review e nel fornire informazioni sulla severità dei difetti.

Indipendentemente dal tipo di review da eseguire, al Technical Test Analyst deve essere concesso un tempo adeguato alla preparazione. Questo include il tempo per eseguire la review del prodotto di lavoro, il tempo per controllare la documentazione referenziata per verificarne la coerenza, e il tempo per determinare cosa potrebbe essere mancante nel prodotto di lavoro. Senza un tempo di preparazione adeguato, la review può diventare un esercizio di editing piuttosto che una vera review. Una buona review include la comprensione di ciò che è scritto, il determinare cosa è mancante, la verifica della correttezza rispetto ad aspetti tecnici, e la verifica che il prodotto descritto è consistente con altri prodotti già sviluppati o in fase di sviluppo. Ad esempio, durante la review di un Test Plan per il livello di test di integrazione, il Technical Test Analyst deve prendere in considerazione anche gli elementi che saranno integrati. Sono pronti per l'integrazione? Esistono dipendenze che devono essere documentate? Sono disponibili i dati per testare i punti di integrazione? Una review non è isolata ai soli prodotti di lavoro che vengono sottoposti a review. Deve anche considerare l'interazione di quell'elemento con gli altri elementi nel sistema.

5.2 Utilizzo delle Checklist nelle Review

Le checklist vengono utilizzate durante le review per ricordare ai partecipanti di verificare punti specifici. Le checklist possono anche aiutare a non personalizzare la review, ad es. "questa è la stessa checklist che utilizziamo per ogni review e non la stiamo utilizzando solo per la review del tuo prodotto di lavoro". Le checklist possono essere generiche e utilizzate per tutte le review o focalizzate su specifiche caratteristiche o aree di qualità. Ad esempio, una checklist generica potrebbe verificare l'utilizzo appropriato dei termini "deve" e "dovrebbe", verificare la formattazione appropriata e gli elementi di conformità similari. Una checklist potrebbe concentrarsi su problemi di sicurezza o problemi di efficienza delle prestazioni.

Le checklist più utili sono quelle sviluppate gradualmente da una singola organizzazione, perché riflettono:

- La natura del prodotto
- L'ambiente di sviluppo locale
  - Staff
  - Strumenti
  - Priorità
- La storia dei precedenti successi e difetti
- Problemi particolari (ad es. efficienza delle prestazioni, sicurezza)

Le checklist dovrebbero essere personalizzate (customizzate) in base all'organizzazione e talvolta al progetto particolare. Le checklist in questo capitolo sono esempi.

5.2.1 Review Architetturale

L'architettura software consiste nei concetti o proprietà fondamentali di un sistema, incorporati nei suoi elementi, nelle relazioni e nei principi della sua progettazione ed evoluzione. [ISO42010].

Le checklist[1] utilizzate per le review architetturali del comportamento nel tempo dei siti web potrebbero, ad esempio, includere la verifica di un'implementazione appropriata dei seguenti elementi, che sono citati da [Web-2]:

- "Connection pooling - Riduzione dell'overhead del tempo di esecuzione associato alla creazione di connessioni del database, stabilendo un pool condiviso di connessioni
- Load balancing - Distribuzione uniforme del carico tra un insieme di risorse
- Distributed processing - Elaborazione distribuita
- Caching - Utilizzo di una copia locale dei dati per ridurre il tempo di accesso
- Lazy instantiation
- Transaction concurrency - Concorrenza delle transazioni
- Isolamento dei processi tra Online Transactional Processing (OLTP) e Online Analytical Processing (OLAP)
- Replica dei dati"

5.2.2 Code Review

Le checklist per le code review sono necessariamente di basso livello e sono più utili quando sono specifiche del linguaggio di programmazione. L'inclusione di anti-pattern a livello di codice è utile, in particolare per gli sviluppatori software meno esperti.

Le checklist[1] utilizzate per le code review potrebbero includere i seguenti elementi:

1. Struttura
   - Il codice implementa completamente e correttamente la progettazione?
   - Il codice è conforme agli standard di codifica pertinenti?
   - Il codice è ben strutturato, consistente nello stile e formattato in modo consistente?
   - Esistono procedure non richiamate o non necessarie, oppure codice irraggiungibile?
   - Esistono stub o routine di test presenti nel codice?
   - È possibile sostituire parti di codice con chiamate a componenti esterne riutilizzabili o funzioni di libreria?
   - Esistono blocchi di codice ripetuto che potrebbero essere condensati in un'unica procedura?
   - L'uso della memoria è efficiente?
   - Vengono utilizzati simboli anziché constanti "numeri magici" o costanti stringa?
   - Esistono moduli eccessivamente complessi che dovrebbero essere ristrutturati o suddivisi in più moduli?

2. Documentazione
   - Il codice è chiaramente e adeguatamente documentato con un formato dei commenti facile da manutenere?
   - Tutti i commenti sono consistenti con il codice?
   - La documentazione è conforme agli standard applicabili?

3. Variabili
   - Tutte le variabili sono definite in modo appropriato con nomi significativi, consistenti e chiari?
   - Esistono variabili ridondanti o inutilizzate?

[1] La domanda d'esame fornirà un sottoinsieme della checklist con cui rispondere alla domanda

4. Operazioni aritmetiche
   - Il codice evita di confrontare l'uguaglianza tra numeri in virgola mobile (floating-point)?
   - Il codice previene sistematicamente errori di arrotondamento?
   - Il codice evita addizioni e sottrazioni su numeri con ordini di grandezza molto differenti?
   - I divisori sono testati con il valore zero o valori spuri?

5. Cicli e rami
   - Tutti i cicli, rami e costrutti logici sono completi, corretti e annidati in modo appropriato?
   - I casi più comuni vengono testati prima nelle catene IF-ELSEIF?
   - Tutti i casi sono coperti in un blocco IF-ELSEIF o CASE, comprese le clausole ELSE o DEFAULT?
   - Ogni istruzione case ha un valore di default?
   - Le condizioni di terminazione del ciclo sono ovvie e sempre raggiungibili?
   - Gli indici o i pedici sono inizializzati in modo appropriato, appena prima del ciclo?
   - Le istruzioni racchiuse all'interno di cicli possono essere posizionate all'esterno dei cicli?
   - Il codice nel ciclo evita di manipolare la variabile indice o di usarla all'uscita del ciclo?

6. Defensive programming
   - Gli indici, i puntatori e i pedici vengono testati rispetto ai limiti di array, record o file?
   - I dati importati e gli argomenti di input vengono testati per verificarne la validità e la completezza?
   - Vengono assegnate tutte le variabili di output?
   - Viene utilizzato il dato corretto in ciascuna istruzione?
   - Viene rilasciata ogni allocazione di memoria?
   - Vengono utilizzati timeout o trap di errore per l'accesso a dispositivi esterni?
   - Viene verificata l'esistenza di un file prima di tentare di accedervi?
   - Tutti i file e i dispositivi vengono lasciati nello stato corretto al termine del programma?

6. Strumenti di Test e Test Automation - 180 minuti

Parole chiave

cattura/riesecuzione, disseminazione dei guasti, emulatore, esecuzione dei test, fault injection, simulatore, testing data-driven, testing keyword-driven, testing model-based (MBT)

Obiettivi di Apprendimento per gli Strumenti di Test e Test Automation

6.1 Definizione del Progetto di Test Automation

TTA-6.1.1 (K2) Riassumere le attività che il Technical Test Analyst esegue durante il set-up di un progetto di test automation
TTA-6.1.2 (K2) Riassumere le differenze tra l'automazione data-driven e l'automazione keyword-driven
TTA-6.1.3 (K2) Riassumere i problemi tecnici più comuni che impediscono ai progetti di test automation di raggiungere il ritorno sull'investimento pianificato
TTA-6.1.4 (K3) Costruire keyword in base a un determinato processo di business

6.2 Strumenti Specifici di Test

TTA-6.2.1 (K2) Riassumere lo scopo degli strumenti per la disseminazione dei guasti e il fault injection
TTA-6.2.2 (K2) Riassumere le principali caratteristiche e i problemi di implementazione per gli strumenti di performance test
TTA-6.2.3 (K2) Spiegare lo scopo generale degli strumenti utilizzati per il testing web-based
TTA-6.2.4 (K2) Spiegare come gli strumenti supportano la pratica del testing model-based
TTA-6.2.5 (K2) Delineare lo scopo degli strumenti utilizzati per supportare il testing di componente e il processo di build
TTA-6.2.6 (K2) Delineare lo scopo degli strumenti utilizzati per supportare il testing delle applicazioni mobile

6.1 Definizione del Progetto di Test Automation

Per essere convenienti, gli strumenti di test (e in particolare quelli che supportano l'esecuzione dei test) devono essere architettati e progettati con cura. L'implementazione di una strategia di test automation senza una solida architettura si traduce solitamente in un insieme di strumenti che sono costosi da manutenere, che sono insufficienti per lo scopo, e non sono in grado di raggiungere il ritorno atteso sull'investimento.

Un progetto di test automation dovrebbe essere considerato come un progetto di sviluppo software. Questo implica la necessità di documentare l'architettura e la progettazione di dettaglio, effettuare le review della progettazione e del codice, svolgere il testing di componente, il testing di integrazione dei componenti e il testing di sistema. Se si usa un codice di test automation instabile o impreciso l'esecuzione dei test può essere inutilmente ritardata o complicata.

Esistono molte attività che il Technical Test Analyst può eseguire per quanto riguarda il test automation:

- Determinare chi sarà responsabile dell'esecuzione dei test automatizzata (eventualmente coordinandosi con un Test Manager).
- Selezionare lo strumento appropriato in base all'organizzazione, alla tempistica, alle competenze del team e ai requisiti di manutenzione (si noti che questo potrebbe implicare la necessità di decidere di creare uno strumento da utilizzare piuttosto che acquisirne uno).
- Definire i requisiti di interfaccia tra lo strumento di test automation e altri strumenti, come gli strumenti di test management e di defect management e gli strumenti utilizzati per il continuous integration.
- Sviluppare adapter, che sono richiesti per creare un'interfaccia tra lo strumento di esecuzione dei test e il software sotto test.
- Selezionare l'approccio di test automation, ad es. keyword-driven o data-driven (si veda il paragrafo 6.1.1).
- Collaborare con il Test Manager per stimare il costo dell'implementazione, inclusa la formazione. Nello sviluppo software Agile questo aspetto verrebbe tipicamente discusso e concordato durante gli sprint/iteration planning meeting con il team completo.
- Schedulare il progetto di test automation e allocare il tempo per la manutenzione.
- Fare formazione ai Test Analyst e ai Business Analyst per l'utilizzo e l'inserimento dei dati per l'automazione.
- Determinare come e quando verranno eseguiti i test automatizzati.
- Determinare come i risultati dei test automatizzati verranno combinati con i risultati dei test manuali.

Nei progetti con una forte enfasi sul test automation, molte di queste attività possono essere affidate a un Test Automation Engineer (per i dettagli si veda il Syllabus Test Automation Engineer [CTSL_TAE_SYL]). Alcune attività organizzative possono essere prese in carico da un Test Manager, in base alle esigenze e alle preferenze di progetto. Nello sviluppo software Agile l'assegnazione di queste attività ai ruoli è tipicamente più flessibile e meno formale.

Queste attività e le decisioni che ne derivano influenzano la scalabilità e la manutenibilità della soluzione di automazione. È necessario dedicare tempo sufficiente alla ricerca delle opzioni, allo studio degli strumenti e delle tecnologie disponibili, e alla comprensione dei piani futuri dell'organizzazione.

6.1.1 Selezione dell'Approccio di Automazione

Questo paragrafo prende in considerazione i seguenti fattori che impattano sull'approccio del test automation:

- Automatizzazione attraverso la GUI, le API e la CLI

- Applicazione di un approccio data-driven
- Applicazione di un approccio keyword-driven
- Gestione dei failure software
- Gestione dello stato del sistema

Il Syllabus Test Automation Engineer [CTSL_TAE_SYL] include ulteriori dettagli sulla selezione di un approccio di automazione.

Automazione attraverso la GUI, le API e la CLI

Il test automation non si limita al testing attraverso la GUI. Esistono anche strumenti per aiutare ad automatizzare il testing a livello di API, attraverso una una Command-Line Interface (CLI) e altri punti di interfaccia nel software sotto test. Una delle prime decisioni che deve prendere il Technical Test Analyst è determinare l'interfaccia più efficace a cui accedere per automatizzare i test. Gli strumenti generali di esecuzione dei test richiedono lo sviluppo di adattatori per queste interfacce. La pianificazione deve quindi considerare l'effort per lo sviluppo dell'adattatore.

Una delle difficoltà del testing attraverso la GUI è la tendenza della GUI a modificarsi man mano che il software evolve. A seconda del modo in cui è progettato il codice di test automation, questo può comportare un notevole onere di manutenzione. Ad esempio, l'utilizzo di funzionalità di cattura/riesecuzione di uno strumento di test automation può produrre test case automatizzati (spesso chiamati test script) che non vengono più eseguiti come ci si aspetta se la GUI cambia. Questo perché il test script registrato cattura le interazioni con gli oggetti grafici quando il tester esegue manualmente il software. Se gli oggetti a cui si accede cambiano, può essere necessario aggiornare anche i test script registrati per riflettere queste modifiche.

Gli strumenti di cattura/riesecuzione possono essere utilizzati come punti di partenza utili per lo sviluppo di test script di automazione. Il tester registra una sessione di test e il test script registrato viene poi modificato per migliorarne la manutenibilità (ad es. sostituendo parte del test script registrato con funzioni riutilizzabili).

Applicazione di un Approccio Data-driven

In base al software da testare, i dati utilizzati per ogni test possono essere differenti, sebbene i passi del test siano virtualmente identici (ad es. testare la gestione degli errori di un campo di input inserendo più valori invalidi e verificando che per ognuno venga restituito l'errore). Non è efficiente sviluppare e mantenere un test script automatizzato per ognuno di questi valori. Una soluzione tecnica comune per questo problema è di spostare i dati dagli script a un archivio esterno come un foglio di calcolo o un database. Vengono scritte le funzioni per accedere ai dati specifici per ogni esecuzione del test script, e questo consente a un singolo test script di utilizzare un insieme di dati di test che specificano i valori di input e i valori dei risultati attesi (ad es. un valore visualizzato in un campo di testo o un messaggio di errore). Questo approccio è chiamato data-driven.

Quando si utilizza questo approccio, in aggiunta ai test script che elaborano i dati forniti, sono necessari un test harness e un'infrastruttura per supportare l'esecuzione dello script o dell'insieme di script. I dati effettivi contenuti nel foglio di calcolo o nel database sono creati dai Test Analyst che hanno familiarità con la funzione di business del software. Nello sviluppo software Agile, può anche essere coinvolto il rappresentante aziendale (ad es. Product Owner) nella definizione dei dati, in particolare per i test di accettazione. Questa suddivisione del lavoro consente ai responsabili dello sviluppo dei test script (ad es. il Technical Test Analyst) di focalizzarsi sull'implementazione dei test script di automazione, mentre il Test Analyst mantiene l'ownership dei test effettivi. Nella maggior parte dei casi, il Test Analyst sarà responsabile dell'esecuzione dei test script una volta che l'automazione è stata implementata e testata.

Applicazione di un Approccio Keyword-driven

Un altro approccio, denominato keyword-driven o action word-driven, fa un ulteriore passo avanti separando dal test script anche l'azione da eseguire sui dati di test forniti [Buwalda01]. Per realizzare questa ulteriore separazione, viene creato un linguaggio di alto livello che è descrittivo e non direttamente eseguibile. Le keyword possono essere definite sia per azioni di alto livello sia per azioni

di basso livello. Ad esempio, le keyword di un processo di business potrebbero includere "Login", "CreateUser", e "DeleteUser". Tali keyword descrivono azioni di alto livello che saranno eseguite nel dominio applicativo. Possono essere usate azioni di livello più basso che denotano l'interazione con l'interfaccia software stessa. Keyword come: “ClickButton”, “SelectFromList”, o “TraverseTree” possono essere utilizzate per testare le funzionalità della GUI che non si adattano perfettamente alle keyword di un processo di business. Le keyword possono contenere parametri, ad esempio la keyword “Login” potrebbe avere due parametri: username e password.

Una volta che sono state definite le keyword e i dati da utilizzare, il test automator (ad es. il Technical Test Analyst o il Test Automation Engineer) traduce le keyword del processo di business e le azioni di livello più basso in codice di test automation. Le keyword e le azioni, insieme ai dati da utilizzare, possono essere memorizzate in fogli di calcolo, oppure essere inserite in strumenti specifici che supportano il test automation keyword-driven. Il framework di test automation implementa la keyword come un insieme di una o più funzioni o test script eseguibili. Gli strumenti leggono i test case scritti con le keyword e richiamano le appropriate funzioni o test script che li implementano. Gli eseguibili sono implementati in modo altamente modulare per consentire un facile mapping a keyword specifiche. Per implementare questi script modulari sono necessarie competenze di programmazione.

Questa separazione della conoscenza logica di business dalla programmazione effettiva richiesta per implementare gli script di test automation, fornisce un uso più efficace delle risorse di test. Il Technical Test Analyst, nel ruolo di test automator, può applicare efficacemente le competenze di programmazione senza dover diventare un esperto di dominio di molte aree di business.

Separare il codice dai dati modificabili aiuta a isolare l'automazione dalle modifiche, migliorando la manutenibilità complessiva del codice e migliorando il ritorno sull'investimento dell’automazione.

Gestione dei Failure Software

Nella progettazione del test automation è importante anticipare e gestire i failure software. Se si verifica un failure, il test automator deve determinare cosa dovrebbe fare il software che il test sta eseguendo. Il failure dovrebbe essere registrato e i test continuano? I test dovrebbero essere terminati? Il failure può essere gestito con un'azione specifica (come il clic di un bottone in una dialog box) o magari aggiungendo un ritardo nel test? I failure software non gestiti possono corrompere i risultati dei test successivi e causare un problema con il test che era in esecuzione quando si è verificato il failure.

Considerare lo Stato del Sistema

Un’altra cosa importante è prendere in considerazione lo stato del sistema all'inizio e alla fine di ogni test. Può essere necessario assicurarsi che il sistema venga riportato a uno stato predefinito dopo il completamento dell'esecuzione dei test. Questo consentirà di eseguire ripetutamente una suite di test automatizzati senza l’intervento manuale che ripristini il sistema a uno stato conosciuto. Per fare questo, il test automation potrebbe, ad esempio, dover eliminare i dati che ha creato o modificare lo stato dei record in un database. Il framework di automazione dovrebbe garantire alla fine dei test che sia stata eseguita una terminazione appropriata (cioè, venga eseguito un logout dopo il completamento dei test).

6.1.2 Modellare i Processi di Business per l’Automazione

Per implementare un approccio keyword-driven per il test automation, i processi di business da testare devono essere modellati nel linguaggio delle keyword di alto livello. È importante che il linguaggio sia intuitivo per gli utenti, che probabilmente sono i Test Analyst che lavorano nel progetto o, nel caso di sviluppo software Agile, il rappresentante di business (ad es. Product Owner).

Le keyword vengono generalmente utilizzate per rappresentare interazioni di alto livello con un sistema. Ad esempio, “Cancel_Order” può richiedere la verifica dell'esistenza dell'ordine, la verifica dei diritti di accesso della persona che richiede la cancellazione, la visualizzazione dell'ordine da cancellare e la conferma della richiesta di annullamento. Le sequenze di keyword (ad es. "Login", "Select_Order", "Cancel_Order") e i dati di test rilevanti vengono utilizzati dal Test Analyst per specificare i test case.

I problemi che devono essere presi in considerazione includono:

- Più dettagliate sono le keyword, più specifici sono gli scenari che possono essere coperti, ma il linguaggio di alto livello può diventare più complesso da manutenere.
- Consentire ai Test Analyst di specificare azioni di basso livello ("ClickButton", "SelectFromList", ecc.) rende i test delle keyword molto più adatti a gestire diverse situazioni. Tuttavia, poiché queste azioni sono correlate direttamente alla GUI, questo potrebbe causare una maggiore manutenzione dei test quando si verificano cambiamenti.
- L'utilizzo di keyword aggregate può semplificare lo sviluppo ma complicare la manutenzione. Ad esempio, possono esistere sei keyword che insieme creano un record. Comunque, creare una keyword di alto livello che chiama tutte le sei le keyword potrebbe in generale non essere l'approccio più efficiente.
- Non importa quanta analisi sia necessaria per il linguaggio delle keyword, esisteranno spesso momenti in cui saranno necessarie keyword nuove e modificate. Esistono due domini separati per una keyword (cioè, la logica di business che sta dietro e la funzionalità di automazione per eseguirla). Pertanto, deve essere creato un processo per gestire entrambi i domini.

Il test automation keyword-driven può ridurre in modo significativo i costi di manutenzione del test automation. Può essere più costoso inizialmente il set-up, ma probabilmente nel suo complesso è più economico se il progetto dura abbastanza a lungo.

Il Syllabus Test Automation Engineer [CTSL_TAE_SYL] include ulteriori dettagli sulla modellizzazione dei processi di business per l'automazione.

6.2 Strumenti Specifici di Test

Questo paragrafo contiene una overview generale degli strumenti che sono probabilmente utilizzati da un Technical Test Analyst, oltre a quanto discusso nel Syllabus Foundation Level [ISTQB_FL_SYL].

Si noti che informazioni dettagliate sugli strumenti sono fornite dai seguenti Syllabi ISTQB®:

- Mobile Application Testing [CTSL_MAT_SYL]
- Performance Testing [CTSL_PT_SYL]
- Model-Based Testing [CTSL_MBT_SYL]
- Test Automation Engineer [CTSL_TAE_SYL]

6.2.1 Strumenti di Disseminazione dei Guasti

Gli strumenti per la disseminazione dei guasti (fault seeding) modificano il codice sotto test (utilizzando se possibile algoritmi predefiniti) per verificare la copertura ottenuta dai test specificati. Quando applicata in modo sistematico, questa attività consente di valutare e, ove necessario, migliorare la qualità dei test (cioè la loro capacità di rilevare i difetti inseriti).

Gli strumenti per la disseminazione dei guasti sono generalmente utilizzati dal Technical Test Analyst ma possono anche essere utilizzati dallo sviluppatore quando esegue il testing di nuovo codice.

6.2.2 Strumenti di Fault Injection

Gli strumenti di fault injection forniscono deliberatamente input errati al software per garantire che il software sia in grado di gestire il difetto. Gli input iniettati causano condizioni negative, che dovrebbero causare l'esecuzione (e il testing) della gestione dell'errore. Questa interruzione del normale flusso di esecuzione del codice aumenta la copertura del codice.

Gli strumenti di fault injection sono generalmente utilizzati dal Technical Test Analyst, ma possono anche essere utilizzati dagli sviluppatori quando esegue il testing di nuovo codice.

6.2.3 Strumenti di Performance Test

Gli strumenti di performance test eseguono le seguenti funzionalità principali:

- Generare carico
- Fornire misurazioni, monitoraggio, visualizzazione e analisi della risposta del sistema a uno specifico carico
- Fornire dettagli sul comportamento delle risorse di sistema e delle componenti di rete

La generazione del carico viene eseguita implementando un profilo operativo predefinito come test script (si veda il paragrafo 4.5.3). Il test script può essere inizialmente catturato per un singolo utente (utilizzando possibilmente uno strumento di cattura/riesecuzione) e può quindi essere implementato per il profilo operativo specificato, utilizzando lo strumento di performance test. Questa implementazione deve tenere conto delle possibili variazioni dei dati per transazione (o insieme di transazioni).

Gli strumenti di performance test generano il carico simulando un gran numero di diversi utenti (utenti "virtuali"), con profili operativi che sono stati definiti per eseguire le attività, incluse la generazione di volumi specifici dei dati di input. Rispetto ai singoli script di test automation, molti script di performance test riproducono l'interazione utente con il sistema a livello di protocollo di comunicazione, e non simulando l'interazione dell'utente tramite una Graphical User Interface (GUI). Questo di solito riduce il numero di "sessioni" separate necessarie durante il testing. Alcuni strumenti di generazione del carico possono anche guidare l'applicazione, utilizzando la sua interfaccia utente per misurare in modo più preciso i tempi di risposta mentre il sistema è sotto carico.

Uno strumento di performance test esegue una vasta gamma di misurazioni, per consentire l'analisi durante o dopo l'esecuzione dei test. Le metriche generate e i report forniti sono in genere:

- Numero di utenti simulati durante il testing
- Numero e tipo di transazioni generate dagli utenti simulati e frequenza di arrivo delle transazioni
- Tempi di risposta a particolari richieste transazionali eseguite dagli utenti
- Report e grafici di carico rispetto ai tempi di risposta
- Report sull'utilizzo delle risorse (ad es. utilizzo nel tempo con valori minimi e massimi)

I fattori più significativi da prendere in considerazione nell'implementazione degli strumenti di performance test includono:

- Hardware e larghezza di banda della rete richiesti per generare il carico
- Compatibilità dello strumento con il protocollo di comunicazione utilizzato dal sistema sotto test
- Flessibilità dello strumento per consentire una facile implementazione di diversi profili operativi
- Funzionalità di monitoraggio, analisi e reportistica richiesti

Gli strumenti di performance test vengono generalmente acquistati piuttosto che sviluppati in-house (internamente), a causa dell'effort di sviluppo richiesto. Tuttavia, può essere appropriato sviluppare uno strumento specifico di performance test, se le restrizioni tecniche impediscono l'utilizzo di un prodotto disponibile, oppure se il profilo di carico e le funzionalità da fornire sono semplici rispetto alle funzionalità e al profilo di carico forniti dagli strumenti commerciali. Ulteriori dettagli sugli strumenti di performance test sono forniti nel Syllabus Performance Testing [CTSL_PT_SYL].

6.2.4 Strumenti per il Testing di Siti Web

Per il testing di siti web sono disponibili diversi strumenti specializzati open source e commerciali. Il seguente elenco mostra lo scopo di alcuni strumenti di test web-based più comuni:

- Strumenti di test degli hyperlink, usati per eseguire la scansione e verificare che in un sito web non siano presenti hyperlink interrotti o mancanti.
- Strumenti di controllo HTML e XML, che verificano la conformità agli standard HTML e XML delle pagine che vengono create per un sito web.

- Strumenti di performance test, per verificare come reagirà il server quando un gran numero di utenti si connettono.
- Strumenti di automazione 'lightweight', che operano con diversi browser.
- Strumenti per la scansione del codice del server, che verificano la presenza di file orfani (non collegati) che sono stati acceduti in precedenza dal sito web.
- Correttori ortografici specifici di HTML.
- Strumenti di controllo di CSS (Cascading Style Sheet).
- Strumenti per verificare la presenza di violazioni degli standard, ad esempio standard di accessibilità Section 508 negli Stati Uniti o M/376 in Europa.
- Strumenti che rilevano diversi problemi di sicurezza.

Le seguenti organizzazioni sono buone fonti per strumenti di test open source:

- World Wide Web Consortium (W3C) [Web-3]. Questa organizzazione definisce gli standard generali per Internet e fornisce diversi strumenti per verificare la presenza di errori rispetto a tali standard.
- Web Hypertext Application Technology Working Group (WHATWG) [Web-5]. Questa organizzazione definisce gli standard HTML e fornisce uno strumento che esegue la validazione dell'HTML [Web-6].

Alcuni strumenti che includono un web spider engine possono anche essere in grado di fornire informazioni sulla dimensione delle pagine e sul tempo necessario per eseguirne il download, oppure sulla presenza o meno di una pagina (ad es. Errore HTTP 404). Questo fornisce informazioni utili per lo sviluppatore, il webmaster e il tester.

I Test Analyst e i Technical Test Analyst utilizzano questi strumenti principalmente durante il testing di sistema.

6.2.5 Strumenti per Supportare il Testing Model-Based

Il testing Model-Based (MBT, Model-Based Testing) è una tecnica dove un modello, come una macchina a stati finiti, viene utilizzato per descrivere il comportamento atteso nel tempo di esecuzione di un sistema software-controlled. Gli strumenti commerciali di MBT (si veda [Utting07]) spesso forniscono un engine che consente a un utente di "eseguire" il modello. I thread di esecuzione interessanti possono essere salvati e utilizzati come test case. MBT è supportato anche da altri modelli eseguibili come le Reti di Petri e gli statechart.

I modelli (e gli strumenti) di MBT possono essere utilizzati per generare grandi insiemi di thread di esecuzione distinti. Gli strumenti di MBT possono aiutare a ridurre l'elevato numero di possibili cammini che possono essere generati in un modello. Utilizzando questi strumenti, il testing può fornire una visione diversa del software da testare. Questo può portare alla scoperta di difetti che potrebbero essere stati dimenticati dal testing funzionale.

Ulteriori dettagli sugli strumenti di test model-based sono forniti nel Syllabus Model-Based Testing [CTSL_MBT_SYL].

6.2.6 Strumenti per le Build e per il Testing di Componente

Anche se gli strumenti per il testing di componente e per l'automazione delle build sono strumenti degli sviluppatori, in molti casi vengono utilizzati e gestiti dai Technical Test Analyst, soprattutto nel contesto dello sviluppo software Agile.

Gli strumenti per il testing di componente sono spesso specifici al linguaggio utilizzato per programmare un componente. Ad esempio, se viene utilizzato Java come linguaggio di programmazione, JUnit potrebbe essere usato per automatizzare il testing di componente. Molti altri linguaggi hanno il proprio strumento di test specifico, che viene denominato genericamente framework xUnit. Tale framework

genera oggetti di test per ogni classe che viene creata, semplificando così le attività che il programmatore deve fare quando automatizza il testing di componente.

Gli strumenti di automazione delle build consentono di attivare automaticamente una nuova build quando viene modificato un componente. Dopo aver completato la build, altri strumenti eseguono automaticamente il testing di componente. Questo livello di automazione per il processo di build è presente in ambienti di continuous integration.

Quando impostato correttamente, questo insieme di strumenti può avere un effetto positivo sulla qualità delle build da rilasciare al testing. Se una modifica apportata da un programmatore introduce difetti di regressione nella build, di solito causerà il fallimento di alcuni dei test automatizzati, attivando un'indagine immediata sulla causa dei failure prima che la build venga rilasciata nell'ambiente di test.

6.2.7 Strumenti per il Supporto del Testing di Applicazioni Mobile

Gli strumenti utilizzati più frequentemente per supportare il testing di applicazioni mobile sono i simulatori e gli emulatori.

Simulatori

Un simulatore di applicazioni mobile modella l'ambiente di esecuzione della piattaforma mobile. Le applicazioni testate su un simulatore vengono compilate in una versione dedicata, che funziona nel simulatore ma non su un dispositivo reale. I simulatori vengono utilizzati come sostituti dei dispositivi reali, ma sono in genere limitati al testing funzionale iniziale e alla simulazione di molti utenti virtuali nel testing di carico. I simulatori sono relativamente semplici (comparati agli emulatori) e possono eseguire i test più velocemente di un emulatore. Tuttavia, l'applicazione testata su un simulatore è diversa dall'applicazione che verrà distribuita.

Emulatori

Un emulatore di applicazioni mobile modella l'hardware e utilizza lo stesso ambiente di esecuzione dell'hardware fisico. Le applicazioni compilate per essere distribuite e testate su un emulatore potrebbero essere utilizzate anche dal dispositivo reale.

Tuttavia, un emulatore non può sostituire completamente un dispositivo perché l'emulatore può comportarsi in modo differente dal dispositivo mobile che tenta di imitare. Inoltre, alcune funzionalità come il (multi)touch e l'accelerometro possono non essere supportate. Questo è in parte causato dalle limitazioni della piattaforma utilizzata per eseguire l'emulatore.

Aspetti Comuni

I simulatori e gli emulatori sono spesso utilizzati per ridurre il costo degli ambienti di test, sostituendo i dispositivi reali. I simulatori e gli emulatori sono utili nella fase iniziale dello sviluppo, poiché questi si integrano in genere con gli ambienti di sviluppo e consentono un rilascio, un testing e un monitoraggio veloce delle applicazioni. L'utilizzo di un emulatore o simulatore ne richiede l'attivazione, l'installazione della app necessaria e il testing della app come se fosse sul dispositivo reale. L'ambiente di sviluppo di ogni sistema operativo mobile è in genere fornito del proprio emulatore o simulatore di serie. Sono disponibili anche emulatori e simulatori di terze parti.

Gli emulatori e i simulatori consentono solitamente di impostare diversi parametri di utilizzo. Queste impostazioni potrebbero includere l'emulazione della rete a velocità differenti, l'intensità del segnale, la perdita di pacchetti, la modifica dell'orientamento, la generazione di interrupt e i dati di posizionamento GPS. Alcune di queste impostazioni possono essere molto utili perché possono essere difficili o costose da replicare con dispositivi reali, come la posizione globale GPS o l'intensità del segnale.

Ulteriori dettagli sono inclusi nel Syllabus Mobile Application Testing [CTSL_MAT_SYL].

7. Riferimenti

7.1 Standard

I seguenti standard sono riportati nei capitoli specificati.

[DO-178C]:	DO-178C - Software Considerations in Airborne Systems and Equipment Certification, RTCA/EUROCAE ED12C. 2013.
	Capitolo 2
[ISO9126]	ISO/IEC 9126-1:2001, Software Engineering – Software Product Quality
	Capitolo 4 e Appendice A
[ISO25010]	ISO/IEC 25010:2011, Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) System and software quality models
	Capitoli 1, 4 e Appendice A
[ISO29119]	ISO/IEC/IEEE 29119-4:2015, Software and Systems Engineering - Software Testing - Part 4: Test techniques
	Capitolo 2
[ISO42010]	ISO/IEC/IEEE 42010:2011, Systems and software engineering - Architecture description
	Capitolo 5
[IEC61508]	IEC 61508-5:2010 Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems, Part 5: Examples of methods for the determination of safety integrity levels
	Capitolo 2
[ISO 26262]	ISO 26262-1:2018, Road vehicles — Functional safety, Parts 1 to 12.
	Capitolo 2
[IEC 62443-3-2]	IEC 62443-3-2:2020, Security for industrial automation and control systems - Part 3-2: Security risk assessment for system design
	Capitolo 4.

7.2 Documenti ISTQB®

[ISTQB_AL_TTA_OVIEW]	Advanced Level Overview, Versione 2019
[CTSL_SEC_SYL]	Advanced Level Security Testing Syllabus, Versione 2016
[CTSL_TAE_SYL]	Advanced Level Test Automation Engineer Syllabus, Versione 2017
[CTSL_FL_SYL]	Foundation Level Syllabus, Versione 2018
[CTSL_PT_SYL]	Foundation Level Performance Testing Syllabus, Versione 2018
[ISTQB_MBT_SYL]	Foundation Level Model-Based Testing Syllabus, Versione 2015
[CTSL_ALTM_SYL]	Advanced Level Test Manager Syllabus, Versione 2012
[CTSL_MAT_SYL]	Foundation Level Mobile Application Testing Syllabus, Versione 2019

[ISTQB_GLOSSARY] Glossary of Terms used in Software Testing, Versione 2019

[CTSL_AuT_SYL] Foundation Level Automotive Software Tester, Versione 2018

7.3 Libri e Articoli

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

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

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

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

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

[Utting07] Mark Utting, Bruno Legeard, “Practical Model-Based Testing: A Tools Approach”, Morgan-Kaufmann, 2007, ISBN: 978-0-12-372501-1

[Whittaker04] James Whittaker and Herbert Thompson, “How to Break Software Security”, Pearson / Addison-Wesley, 2004, ISBN 0-321-19433-0

[Wiegers02] Karl Wiegers, "Peer Reviews in Software: A Practical Guide", Addison-Wesley, 2002, ISBN 0-201-73485-0

7.4 Altri Riferimenti

I seguenti riferimenti puntano a informazioni disponibili in Internet. Anche se i riferimenti sono stati verificati al momento della pubblicazione di questo Syllabus, ISTQB® non si assume la responsabilità nel caso in cui i riferimenti non siano più disponibili.

[Web-1] http://www.nist.gov (NIST National Institute of Standards and Technology)

[Web-2] http://www.codeproject.com/KB/architecture/SWArchitectureReview.aspx

[Web-3] http://www.W3C.org

[Web-4] https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

[Web-5] https://whatwg.org

[Web-6] https://validator.w3.org/

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

Capitolo 2: [Web-7]

Capitolo 4: [Web-1] [Web-4]

Capitolo 5: [Web-2]

Capitolo 6: [Web-3] [Web-5] [Web-6]

8. Appendice A: Overview delle Caratteristiche di Qualità

La seguente tabella confronta le caratteristiche di qualità descritte nello standard ISO 9126-1 ora sostituito (usato nella versione 2012 del Syllabus Technical Test Analyst) con quelle del più recente standard ISO25010 (usato nell'ultima versione 2019 del Syllabus).
Si noti che l'adeguatezza funzionale e l'usabilità sono coperte dal Syllabus Test Analyst.

ISO/IEC 25010	ISO/IEC 9126-1	Note
Adeguatezza Funzionale	**Funzionalità**	Il nuovo nome è più accurato, ed Evita confusione con altri significati di "funzionalità"
Completezza funzionale		Copertura delle necessità dichiarate
Correttezza funzionale	Accuratezza	Più generale dell'accuratezza
Appropriatezza funzionale	Adeguatezza	Copertura delle necessità implicite
	Interoperabilità	Spostato in Compatibilità
	Sicurezza	Ora una caratteristica
Efficienza delle Prestazioni	**Efficienza**	Rinominato per evitare conflitti con la definizione di efficienza in ISO/IEC 25062
Comportamento nel tempo	Comportamento nel tempo	
Utilizzo delle risorse	Utilizzo delle risorse	
Capacità		Nuova sotto-caratteristica
Compatibilità		Nuova caratteristica
Coesistenza	Coesistenza	Spostato da Portabilità
Interoperabilità		Spostato da Funzionalità (Test Analyst)
Usabilità		Problema di qualità implicito reso esplicito
Riconoscibilità dell'appropriatezza	Understandability	Il nuovo nome è più accurato
Apprendibilità	Apprendibilità	
Operabilità	Operabilità	
Protezione da errori dell'utente		Nuova sotto-caratteristica
Estetica della user interface	Attrattività	Il nuovo nome è più accurato
Accessibilità		Nuova sotto-caratteristica
Affidabilità	**Affidabilità**	
Maturità	Maturità	
Disponibilità		Nuova sotto-caratteristica
Tolleranza ai guasti	Tolleranza ai guasti	
Recuperabilità	Recuperabilità	
Sicurezza	**Sicurezza**	Nessuna precedente sotto-caratteristica
Confidenzialità		Nessuna precedente sotto-caratteristica
Integrità		Nessuna precedente sotto-caratteristica
Non-repudiation		Nessuna precedente sotto-caratteristica
Responsabilità		Nessuna precedente sotto-caratteristica

Autenticità		Nessuna precedente sotto-caratteristica
Manutenibilità	**Manutenibilità**	
Modularità		Nuova sotto-caratteristica
Riutilizzabilità		Nuova sotto-caratteristica
Analizzabilità	Analizzabilità	
Modificabilità	Stabilità	Nome più accurato che combina modificabilità e stabilità
Testabilità	Testabilità	
Portabilità	**Portabilità**	
Adattabilità	Adattabilità	
Installabilità	Installabilità	
	Coesistenza	Spostato in Compatibilità
Sostituibilità	Sostituibilità	

9. Indice