Certified Tester

Advanced Level Syllabus
Technical Test Analyst

Versione 2019

International Software Testing Qualifications Board

Gruppo di lavoro dell'Advanced Level:

Graham Bath (vicepresidente), Rex Black, Judy McKay, Kenji Onoshi, Mike Smith (presidente), Erik van Veenendaal

Cronologia revisioni

Versione	Data	Indicazioni
2012	19 Ottobre 2012	Rilascio dell'Assemblea Generale per la versione 2012
2019 Alpha	1 Marzo 2019	Versione per la revisione Alpha
2019 Pre-Beta	14 Aprile 2019	Versione contenente i commenti alla revisione Alpha
2019 Beta	6 Maggio 2019	Versione per la revisione Beta
2019 V.10	18 Ottobre 2019	Rilascio dell'Assemblea Generale per la versione 2019

Sommario

Riconoscimenti

Questo documento è stato redatto da un team di professionisti dell'International Software Testing Qualifications Board Advanced Level Working Group: Graham Bath (vicepresidente), Rex Black, Judy McKay, Kenji Onoshi, Mike Smith (presidente), Erik van Veenendaal

Alla revisione, al commento o al voto di questo syllabus hanno partecipato:

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

Il team ringrazia i suddetti partecipanti e i National Board per tutti i suggerimenti e i contributi.

Il presente documento è stato ufficialmente rilasciato dall'Assemblea Generale dell'ISTQB® il 20 Ottobre 2019.

0. Premessa

0.1 Scopo di questo Sillabo

Questo syllabus costituisce la base per la International Software Testing Qualification di Livello Avanzato per i Technical Test Analyst. L'ISTQB® mette a disposizione questo syllabus come segue:

1. Ai National Board, per consentire la traduzione in lingua locale e accreditare gli enti di formazione. I National Board possono adattare il syllabus alle loro esigenze linguistiche particolari e modificare i riferimenti per adattarli alle pubblicazioni locali.
2. Agli Exam Board, per consentire la formulazione delle domande d'esame in lingua locale, adattate agli obiettivi di apprendimento di ciascun syllabus.
3. Agli enti di formazione, per consentire la produzione del materiale didattico e determinare i metodi di insegnamento appropriati.
4. Ai candidati alla certificazione, per la preparazione all'esame (sia se prendono parte a un corso di formazione sia se studiano individualmente).
5. Alla community internazionale di ingegneria del software e dei sistemi, per progredire nella professione di testing di software e sistemi e come base per libri e articoli.

ISTQB® può consentire ad altri organismi di utilizzare il presente syllabus per altri scopi, previa autorizzazione scritta.

0.2 Il Certified Tester Advanced Level nel Software Testing

La qualifica Advanced Level comprende i tre syllabi relativi ai seguenti ruoli:

- Test Manager
- Test Analyst
- Technical Test Analyst

L' ISTQB Advanced Level Overview 2019 è un documento a parte [ISTQB_AL_OVIEW] che include le seguenti informazioni:

- Business Outcome
- Matrice di tracciabilità tra Business Outcome e obiettivi di apprendimento
- Sommario

0.3 Obiettivi di apprendimento verificabili e livelli cognitivi di conoscenza

Gli obiettivi di apprendimento supportano i Business Outcome e sono utilizzati per creare gli esami per il conseguimento della certificazione Advanced Technical Test Analyst.

All'inizio di ogni capitolo sono indicati i livelli di cognitivi di conoscenza degli obiettivi di apprendimento specifici, di livello K2, K3 and K4, classificati come segue:

- K2: Comprensione
- K3: Applicazione
- K4: Analisi

0.4 Esperienza attesa

Alcuni degli obiettivi di apprendimento per il Technical Test Analyst presuppongono esperienza di base nelle seguenti aree:

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

0.5 L'esame per Advanced Level Technical Test Analyst

Questo syllabus costituisce la base per l'esame Advanced Level Technical Test Analyst. Le risposte alle domande d'esame possono richiedere conoscenze basate su più di una sezione di questo syllabus. Tutte le sezioni del syllabus sono argomento d'esame, ad eccezione dell'introduzione e delle appendici. Standard, libri e altri syllabi ISTQB® sono inclusi come riferimenti, ma il loro contenuto non è tema di esame, al di là di quanto riassunto nel presente syllabus.

L'esame si basa su domande a risposta multipla ed è costituito da 45 domande. Per superare l'esame, è necessario rispondere correttamente ad almeno il 65% delle domande.

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

0.6 Requisiti d'ingresso per l'esame

Per l'iscrizione all'esame di certificazione Advanced Level Technical Test Analyst è necessario aver conseguito la certificazione Certified Tester Foundation Level.

0.7 Accreditamento dei corsi

Gli ISTQB Member Board possono accreditare fornitori della formazione (training provider) i cui materiali di training seguono questo syllabus. A questo scopo i fornitori della formazione devono richiedere le linee guida per l'accreditamento al Member Board o all'organizzazione che effettuerà l'accreditamento. Un corso è accreditato se è riconosciuto conforme a questo syllabus, ed è autorizzato ad avere un esame ISTQB all'interno del corso stesso.

0.8 Livello di dettaglio del syllabus

Il livello di dettaglio di questo syllabus consente la conduzione di corsi e esami consistenti a livello internazionale. Per ottenere questo scopo il syllabus consiste di:

- Obiettivi didattici generali indicanti gli intenti dell'Advanced Level Technical Test Analyst
- Un elenco di termini che gli studenti devono essere in grado di ricordare e comprendere
- Obiettivi di apprendimento per ogni area di conoscenza, che illustrano il risultato cognitivo di apprendimento da raggiungere
- Una descrizione dei concetti chiave, compresi I riferimenti a fonti quali letteratura accettata o standard.

I contenuti del syllabus non costituiscono una descrizione dell'intera area di conoscenza; riflettono il livello di dettaglio che deve essere coperto da corsi di Livello Avanzato. Si focalizzano su materiale applicabile a qualunque progetto software utilizzando qualunque ciclo di vita. Il syllabus non include alcun specifico obiettivo di apprendimento legato ad un particolare ciclo o metodo di sviluppo, esaminando tuttavia come i concetti si possano applicare a progetti Agile, ad altri tipi di cicli di sviluppo iterativi e incrementali, e a cicli di sviluppo sequenziali.

0.9 Organizzazione del syllabus

I capitoli con contenuti oggetto di esame sono sei. L'intestazione di alto livello di ogni capitolo specifica il tempo minimo necessario, ma non viene fornita la tempistica a livello di sotto-capitolo. Per corsi accreditati, questo syllabus richiede almeno 21 ore e 15 minuti di lezione, distribuiti come segue:

- Capitolo 1: I Compiti del Technical Test Analyst nel Testing Basato sul Rischio (30 minuti)

- Capitolo 2: Tecniche di Test White-Box (345 minutes)
- Capitolo 3: Tecniche Analitiche (210 minuti)
- Capitolo 4: Caratteristiche di Qualità per il Testing Tecnico (345 minuti)
- Capitolo 5: Review (165 minuti)
- Capitolo 6: Strumenti di Test e Automazione (180 minuti)

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

Parole chiave
rischio di prodotto, 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 nel Testing Basato sul Rischio

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

TTA-1.2.2 (K2) Riepilogare le attività del Technical Test Analyst in un approccio di testing basato sul rischio

1.1 Introduzione

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

I Technical Test Analyst lavorano quindi all'interno del framework di testing basato sul rischio stabilito dal Test Manager per un determinato progetto e contribuiscono con la loro conoscenza dei rischi tecnici del prodotto inerenti al progetto, per esempio i rischi relativi alla sicurezza, all'affidabilità e alle prestazioni.

1.2 Attività nel Testing Basato sul Rischio

In ragione della loro particolare competenza tecnica, i Technical Test Analyst sono attivamente coinvolti nelle seguenti attività di testing basate sul rischio:

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

Queste attività vengono eseguite in modo iterativo durante tutto il progetto per affrontare l'evoluzione dei rischi e delle priorità, e per valutare e comunicare regolarmente lo stato del rischio.

1.2.1 Identificazione del Rischio

Con 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, fare brainstorming con i colleghi e analizzare le esperienze attuali e passate per determinare dove si trovano le probabili aree di rischio. In particolare, i Technical Test Analyst lavorano a stretto contatto con altri stakeholder, sviluppatori, architetti, tecnici delle operations, product owner, uffici di supporto locali e tecnici di service desk, per determinare le aree di rischio tecnico che incidono sul prodotto e sul progetto. Il coinvolgimento di altri stakeholder è generalmente facilitato dai Test Manager per garantire che tutte le opinioni siano prese in considerazione.

I rischi che possono essere identificati dal Technical Test Analyst sono tipicamente basati sulle caratteristiche di qualità [ISO25010] elencate al capitolo 4, e includono ad esempio:

- Efficienza delle prestazioni (ad es. l'incapacità di ottenere i tempi di risposta richiesti in condizioni di carico elevato)
- Sicurezza (ad es. la divulgazione di dati sensibili a causa di attacchi alla sicurezza)
- Affidabilità (ad es. l'applicazione non è in grado di soddisfare la disponibilità specificata nel Service Level Agreement)

1.2.2 Valutazione del rischio

Mentre l'identificazione del rischio riguarda l'identificazione del maggior numero possibile di rischi pertinenti, la valutazione del rischio è lo studio degli stessi al fine di classificare ogni rischio e determinare la probabilità e l'impatto ad esso associati. La probabilità che un rischio si verifichi è generalmente interpretata come la probabilità che il potenziale problema possa esistere nel sistema sottoposto a test.

Il Technical Test Analyst contribuisce a trovare e comprendere il potenziale rischio tecnico del prodotto per ciascun elemento di rischio, mentre il Test Analyst contribuisce a comprendere il potenziale impatto di business del problema nel caso si verifichi.

I rischi di progetto possono influire sul successo complessivo dello stesso. In genere, è necessario considerare i seguenti rischi generici di progetto:

- Conflitto tra gli stakeholder per quanto riguarda i requisiti tecnici
- Problemi di comunicazione derivanti dalla distribuzione geografica dell'organizzazione di sviluppo
- Strumenti e tecnologia (comprese le relative competenze)
- Tempo, risorse e pressioni del management
- Assenza di attività anticipate di quality assurance
- Elevati tassi di variazione dei requisiti tecnici

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

- Complessità della tecnologia
- Complessità della struttura del codice
- Quantità di riutilizzo rispetto al nuovo codice
- Elevato numero di difetti riscontrati relativi alle caratteristiche tecniche di qualità (storico dei difetti)
- Problemi tecnici di interfaccia e integrazione

Date le informazioni di rischio disponibili, il Technical Test Analyst propone un livello di rischio iniziale secondo le linee guida stabilite dal Test Manager. Ad esempio, il Test Manager può stabilire che i rischi dovrebbero essere classificati con un valore compreso tra 1 e 10, dove 1 è il rischio più alto. Il valore iniziale può essere modificato dal Test Manager quando tutte le opinioni degli stakeholder sono state prese in considerazione.

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:

- Riduzione del rischio, grazie all'esecuzione dei test più importanti (quelli che riguardano le aree ad alto rischio) e alla messa in atto di misure appropriate di mitigazione e di gestione della contingenza che seguono quanto indicato nel test plan
- Valutazione dei rischi sulla base di informazioni aggiuntive raccolte durante lo svolgimento del progetto, e utilizzo di tali informazioni per implementare misure di mitigazione volte a ridurre la probabilità o evitare l'impatto di tali rischi

Il Technical Test Analyst collabora spesso con specialisti in aree quali la sicurezza e le prestazioni, per definire le misure di mitigazione del rischio e gli elementi della strategia di test dell'organizzazione. Informazioni aggiuntive possono essere ricavate da syllabi specialistici di ISTQB® quali l'Advanced Level Security Testing [ISTQB_ALSEC_SYL] e il Foundation Level Performance Testing [ISTQB_FLPT_SYL].

2. Tecniche di Test White-box - 345 minuti

Parole chiave

testing delle API, condizione atomica, testing del flusso di controllo, complessità ciclomatica, testing delle decisioni, testing delle decisioni/condizioni modificate, testing delle condizioni multiple, testing dei cammini, cortocircuito, testing delle istruzioni, tecnica di test white-box.

Obiettivi di apprendimento per il Testing White-Box

Nota: Gli obiettivi di apprendimento 2.2.1, 2.3.1, 2.4.1, 2.5.1, 2.6.1 sono riferiti ad un “elemento di specifica” come sezioni di codice, requisiti, user story, use case e specifiche funzionali.

2.2 Testing delle istruzioni

TTA-2.2.1 (K3) Scrivere test case per un determinato elemento di specifica applicando la tecnica del testing delle istruzioni per raggiungere un livello di copertura definito

2.3 Testing delle decisioni

TTA-2.3.1 (K3) Scrivere test case per un determinato elemento di specifica applicando la tecnica del testing delle decisioni per raggiungere un livello di copertura definito

2.4 Testing della copertura delle decisioni / condizioni modificate (MC/DC)

TTA-2.4.1 (K3) Scrivere test case applicando la tecnica del testing della copertura delle decisioni/ condizioni modificate (MC/DC) per raggiungere un livello di copertura definito

2.5 Testing delle condizioni multiple

TTA-2.5.1 (K3) Scrivere test case per un determinato elemento di specifica applicando la tecnica del testing delle condizioni multiple per raggiungere un livello di copertura definito

2.6 Testing dei cammini

TTA-2.6.1 (K3) Scrivere test case per un determinato elemento di specifica applicando il metodo della baseline semplificato di McCabe

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 trovare

2.8 Selezione di una tecnica di Test White-box

TTA-2.8.1 (K4) Selezionare una tecnica di test white-box adatta ad una determinata situazione progettuale

2.1 Introduzione

Questo capitolo descrive principalmente le tecniche di test white-box. Queste tecniche si applicano al codice e ad altre strutture quali i diagrammi di flusso dei processi di business.

Ogni tecnica specifica consente di derivare sistematicamente i test case e si concentra su un aspetto particolare della struttura da considerare. Le tecniche forniscono criteri di copertura che devono essere misurati e associati a un obiettivo definito da ciascun progetto o organizzazione. Il raggiungimento di una copertura completa non significa che l'intera serie di test sia completa, ma piuttosto che la tecnica utilizzata non suggerisce più alcun test utile per la struttura in esame.

In questo syllabus sono prese in considerazione le seguenti tecniche:

- Testing delle istruzioni
- Testing delle decisioni
- Testing della copertura delle decisioni/condizioni modificate (MC/DC)
- Testing delle condizioni multiple
- Testing dei cammini di base
- Testing delle API

Il syllabus Foundation [ISTQB_FL_SYL] introduce il testing delle istruzioni e delle decisioni. Il testing delle istruzioni esercita le istruzioni eseguibili nel codice, mentre il testing delle decisioni esercita le decisioni nel codice e testa il codice che viene eseguito in base ai risultati della decisione.

Le tecniche MC/DC e di testing delle condizioni multiple sono basati su predicati decisionali e trovano in generale gli stessi tipi di difetti. Per quanto complesso possa essere un predicato decisionale, il risultato sarà VERO o FALSO, cosa che determinerà il percorso da intraprendere nel codice. Viene rilevato un difetto quando il percorso previsto non viene intrapreso perché un predicato decisionale non viene valutato come previsto.

Le prime quattro tecniche sono progressivamente più approfondite (e il testing dei cammini base è più completo del testing delle istruzioni e delle decisioni); tecniche più approfondite generalmente richiedono la definizione di un numero maggiore di test per ottenere la copertura prevista e trovare difetti più fini.

Riferimenti: [Bath14], [Beizer90], [Beizer95], [Copeland03], [McCabe96], [ISO29119] e [Koomen06].

2.2 Testing delle istruzioni

Il testing delle istruzioni esercita le istruzioni eseguibili nel codice. La copertura si misura come numero di istruzioni eseguite dai test rispetto al numero totale di istruzioni eseguibili nell'oggetto di test, normalmente espresso in percentuale.

Applicabilità
Questo livello di copertura dovrebbe essere considerato il minimo per tutto il codice in fase di test.

Limitazioni/Difficoltà
Le decisioni non vengono prese in considerazione. Anche alte percentuali di copertura delle istruzioni potrebbero non rilevare alcuni difetti nella logica del codice.

2.3 Testing delle decisioni

Il testing delle decisioni esercita le decisioni nel codice e verifica il codice eseguito in base ai risultati delle decisioni. Per fare ciò, i test case seguono i flussi di controllo che si verificano da un punto di decisione (ad esempio, per un'istruzione IF, uno per il risultato vero e uno per il risultato falso; per un'istruzione CASE, si dovrebbero avere test case per tutti i possibili risultati, incluso il risultato di default). La copertura si misura come numero di risultati decisionali eseguiti dai test rispetto al numero totale di risultati decisionali nell'oggetto del test, normalmente espresso in percentuale.

Rispetto alle tecniche MC/DC e delle condizioni multiple descritte nel seguito, il testing delle decisioni considera l'intera decisione nel suo insieme e valuta i risultati VERO e FALSO attraverso test case separati.

Applicabilità
Questo livello di copertura dovrebbe essere preso in considerazione quando il codice da testare è importante o addirittura critico (vedere la tabella nella Sezione 2.8)

Limitazioni/Difficoltà
Poiché potrebbe richiedere un numero maggiore di test case rispetto al testing solo a livello di istruzione, potrebbe essere problematico quando il tempo è limitato. Il testing delle decisioni non considera i dettagli di come viene presa una decisione con più condizioni e potrebbe non riuscire a rilevare i difetti causati da combinazioni di queste condizioni.

2.4 Testing della copertura delle decisioni/condizioni modificate (MC/DC)

Rispetto al testing delle decisioni, che considera l'intera decisione nel suo insieme e valuta i risultati VERO e FALSO con test case separati, il testing MC/DC considera come viene presa una decisione quando include più condizioni (altrimenti è semplicemente testing delle decisioni).

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

Questa tecnica fornisce un livello di copertura più elevato rispetto alla copertura delle istruzioni o delle decisioni quando ci sono decisioni che contengono più condizioni. Supponendo che ci siano N condizioni atomiche uniche e indipendenti, si può generalmente ottenere l'MC/DC con N+1 test case. MC/DC richiede coppie di test case che mostrano che una singola condizione atomica può influenzare in modo indipendente il risultato di una decisione.

Nell'esempio che segue, prendiamo in considerazione un'istruzione "IF (A OR B) AND C, THEN ...".

	A	B	C	(A or B) and C
Test 1	VERO	FALSO	VERO	VERO
Test 2	FALSO	VERO	VERO	VERO
Test 3	FALSO	FALSO	VERO	FALSO
Test 4	VERO	FALSO	FALSO	FALSO

Nel Test 1, A è VERO e il risultato complessivo è VERO. Se A viene modificato in FALSO (come nel Test 3, mantenendo gli altri valori invariati) il risultato cambia in FALSO, dimostrando così che A può influenzare in modo indipendente il risultato della decisione.
Nel test 2, B è VERO e il risultato complessivo è VERO. Se B viene modificato in FALSO (come nel Test 3, mantenendo gli altri valori invariati) il risultato cambia in FALSO, dimostrando così che B può influenzare in modo indipendente il risultato della decisione.

Nel test 1, C è VERO e il risultato complessivo è VERO. Se C viene modificato in FALSO (come nel Test 4, mantenendo gli altri valori invariati) il risultato cambia in FALSO, dimostrando così che C può influenzare in modo indipendente il risultato della decisione.

Si noti che, a differenza delle tecniche testing delle istruzioni e delle decisioni, non ci sono "livelli di copertura definiti" per MC/DC; o si raggiunge (cioè, 100%) o no.

Applicabilità
Questa tecnica viene utilizzata nell'industria aerospaziale e in altri settori che producono sistemi critici per la safety. Viene utilizzato nel testing di software in cui un guasto può causare una catastrofe.

Limitazioni/Difficoltà
Il raggiungimento della copertura MC/DC può risultare complicato quando ci sono più occorrenze di una variabile in una decisione con più condizioni; quando ciò si verifica, le condizioni possono essere “accoppiate”. A seconda della decisione, potrebbe non essere possibile variare il valore di una condizione in modo tale che da sola provochi il cambiamento del risultato della decisione. Un approccio per affrontare questo problema è specificare che solo le condizioni atomiche disaccoppiate devono essere testate a livello MC/DC. L'altro approccio consiste nell'analizzare caso per caso ogni decisione in cui si verifica l'accoppiamento.

Alcuni linguaggi di programmazione e/o interpreti sono progettati in modo tale da cortocircuitare (ovvero, esibire un comportamento di short-circuiting per) la valutazione di una dichiarazione di decisione complessa nel codice. Cioè, il codice in esecuzione potrebbe non valutare un'intera espressione se il risultato finale della valutazione può essere determinato dopo aver valutato solo una parte dell'espressione. Ad esempio, se si valuta la decisione "A AND B", non c'è motivo di valutare B se A è già stato valutato come FALSO. Nessun valore di B può modificare il risultato finale, quindi il codice potrebbe far risparmiare tempo di esecuzione non valutando B. Questo cortocircuito può influire sulla capacità di raggiungere la copertura MC/DC poiché alcuni test richiesti potrebbero non essere realizzabili.

2.5 Testing delle condizioni multiple

In rari casi, potrebbe essere necessario testare tutte le possibili combinazioni di condizioni atomiche che una decisione può contenere. Questo livello esaustivo di test è chiamato testing delle condizioni multiple. Il numero di test richiesti dipende dal numero di condizioni atomiche nella decisione e può essere determinato calcolando 2^N dove N è il numero di condizioni atomiche disaccoppiate. Utilizzando lo stesso esempio di prima, sono necessari i seguenti test per ottenere la copertura delle condizioni multiple:

	A	B	C	(A or B) and C
Test 1	VERO	VERO	VERO	VERO
Test 2	VERO	VERO	FALSO	FALSO
Test 3	VERO	FALSO	VERO	VERO

Test 4	VERO	FALSO	FALSO	FALSO
Test 5	FALSO	VERO	VERO	VERO
Test 6	FALSO	VERO	FALSO	FALSO
Test 7	FALSO	FALSO	VERO	FALSO
Test 8	FALSO	FALSO	FALSO	FALSO

La copertura si misura come numero di combinazioni di condizioni uniche eseguite dai test rispetto al numero totale di combinazioni di condizioni nell'oggetto di test, normalmente espresso in percentuale.

Applicabilità
Questa tecnica viene utilizzata per testare software embedded che dovrebbe funzionare in modo affidabile senza crash per lunghi periodi di tempo (ad esempio, interruttori telefonici che dovrebbero durare 30 anni).

Limitazioni/Difficoltà
Poiché il numero di test case può essere derivato direttamente da una tabella di verità contenente tutte le condizioni atomiche, questo livello di copertura può essere facilmente determinato. Tuttavia, l'enorme numero di test case richiesti rende più fattibile, nella maggior parte dei casi, la copertura MC/DC.

Se il linguaggio di programmazione applica cortocircuiti (short-circuiting), il numero di test case effettivi sarà spesso ridotto, a seconda dell'ordine e del raggruppamento delle operazioni logiche eseguite sulle condizioni atomiche.

2.6 Testing dei cammini base

Il testing dei cammini consiste in generale nell'identificare i percorsi del codice e nel creare i test per coprirli. Concettualmente sarebbe utile testare ogni percorso univoco, tuttavia, in qualsiasi sistema non banale, il numero di test case potrebbe diventare eccessivamente grande a causa della natura dei cicli. Al contrario, il testing dei cammini base può essere realisticamente eseguito e segue il metodo della baseline semplificato sviluppato da McCabe [McCabe96].

Questa tecnica si applica con i seguenti passaggi:

1. Creazione di un grafo del flusso di controllo per un dato elemento di specifica (ad esempio, codice o specifica di progettazione funzionale). Si noti che questo può anche essere il primo passo per eseguire l'analisi del flusso di controllo (vedi sezione 3.2.1).
2. Selezione di un cammino di base attraverso il codice (non un percorso di eccezione). Questo cammino base (baseline) dovrebbe essere quello più importante da testare: per determinarlo si può usare l'analisi del rischio.
3. Generazione del secondo cammino modificando il risultato della prima decisione sul percorso, mantenendo il numero massimo di risultati decisionali uguale al percorso di base.
4. Generazione del terzo percorso iniziando di nuovo con il cammino base e modificando il risultato della seconda decisione sul percorso. Quando si incontrano decisioni a più vie (ad esempio, un'istruzione case), occorre esercitare ogni risultato della decisione prima di passare alla decisione successiva.
5. Generazione di ulteriori cammini, modificando ciascuno dei risultati sul cammino base. Quando si prendono nuove decisioni, il risultato più importante dovrebbe essere seguito per primo.
6. Una volta che tutti i risultati decisionali sul cammino di base sono stati coperti, applicare lo stesso approccio ai cammini successivi finché tutti i risultati decisionali nell'elemento di specifica non sono stati esercitati.

Applicabilità
Il metodo della baseline semplificato, come definito sopra, viene spesso eseguito su software mission critical. Integra bene gli altri metodi trattati in questo capitolo perché guarda ai percorsi del software invece di limitarsi al modo in cui vengono prese le decisioni.

Limitazioni/Difficoltà
Alla data di rilascio di questo syllabus, il supporto fornito da strumenti di test per il testing dei cammini base è piuttosto limitato.

Copertura
La tecnica sopra descritta garantisce la copertura completa di tutti i percorsi linearmente indipendenti e il numero di cammini corrisponde alla complessità ciclomatica del codice. A seconda della complessità del codice, può essere utile utilizzare uno strumento per verificare che sia stata raggiunta la copertura completa dell'insieme di cammini base. La copertura si misura come numero di percorsi linearmente indipendenti eseguiti dai test rispetto al numero totale di percorsi linearmente indipendenti nell'oggetto di test, normalmente espresso in percentuale. Il test dei cammini base fornisce test più approfonditi rispetto alla copertura delle decisioni, con un incremento relativamente basso del numero di test [NIST96].

2.7 Testing delle API

Le API (Application Programming Interface) sono elementi di codice che consentono la comunicazione tra diversi processi, programmi e/o sistemi. Le API vengono spesso utilizzate in una relazione client/ server in cui un processo fornisce un qualche tipo di funzionalità ad altri processi.

Il testing delle API è un tipo di testing piuttosto che una tecnica. Per certi aspetti, il testing delle API è abbastanza simile al testing di un'interfaccia utente grafica (GUI). L'attenzione si concentra sulla valutazione dei valori di input e dei dati restituiti.

Quando si tratta di API i test negativi sono spesso cruciali. I programmatori che utilizzano le API per accedere a servizi esterni al proprio codice possono provare a utilizzare le interfacce API in modi per cui non erano previste. Ciò significa che è necessaria una gestione degli errori molto robusta, per evitare operazioni errate. Potrebbe essere necessario eseguire test combinatori di molte interfacce diverse 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 molti modi.

Le API spesso sono accoppiate in modo lasco (loosely coupled), e ciò dà luogo alla possibilità di perdere transazioni o a problemi di temporizzazione. Ciò richiede un testing approfondito dei meccanismi di ripristino e ripetizione. Un'organizzazione che fornisce un'interfaccia API deve garantire che tutti i servizi abbiano una disponibilità molto elevata; questo spesso richiede rigorosi test di affidabilità da parte di chi pubblica l'API, nonché supporto infrastrutturale.

Applicabilità
Il testing delle API sta diventando sempre più importante per testare sistemi di sistemi, man mano che i singoli sistemi diventano distribuiti o utilizzano remote processing come metodo per scaricare il carico di lavoro su altri processori. Alcuni esempi sono:

- Chiamate di sistemi operativi
- Service-oriented architecture (SOA)
- Remote procedure call (RPC)
- Web service

La containerizzazione del software [Burns18] si traduce nella suddivisione di un programma software in diversi contenitori che comunicano tra loro utilizzando meccanismi come quelli elencati sopra. Il testing delle API dovrebbe anche indirizzare queste interfacce.

Limitazioni/Difficoltà
Il testing diretto di un'API di solito richiede che un Technical Test Analyst utilizzi strumenti specializzati. Poiché in genere non esiste un'interfaccia grafica diretta associata a un'API, potrebbero essere necessari strumenti per configurare l'ambiente iniziale, fare il marshalling dei dati, invocare l'API e determinare il risultato.

Copertura
Il testing delle API è un tipo di testing e non denota alcun livello specifico di copertura. Come minimo, il testing delle API dovrebbe includere l'esecuzione di chiamate con valori di input realistici e input imprevisti per controllare la gestione delle eccezioni. Test delle API più approfonditi possono garantire che le entità richiamabili vengano esercitate almeno una volta o che tutte le chiamate possibili vengano effettuate almeno una volta.

Tipi di difetti
I tipi di difetti che possono essere trovati testando le API sono piuttosto disparati. I problemi di interfaccia sono comuni, così come i problemi di gestione dei dati, i problemi di temporizzazione, la perdita e la duplicazione delle transazioni.

2.8 Selezione di una tecnica di Test White-box

Il contesto del sistema sottoposto a test ha un impatto sui livelli di rischio e criticità del prodotto (vedi sotto). Questi fattori influenzano la metrica di copertura richiesta (e quindi la tecnica di test white-box da utilizzare) e la profondità della copertura da raggiungere. In generale, più critico è il sistema e più alto è il livello di rischio del prodotto, più rigorosi sono i requisiti di copertura e maggiore è la necessità di tempo e risorse per ottenere la copertura desiderata.

A volte la metrica di copertura richiesta può essere derivata da standard applicabili al sistema software in questione. Ad esempio, nel caso di un software per il settore aeronautico, potrebbe essere richiesta la conformità allo standard DO-178C (in Europa, ED-12C.) che contiene le seguenti cinque condizioni di failure:

A. Catastrofico (Catastrophic): failure che può causare la mancanza della funzione critica necessaria per far volare o atterrare in sicurezza l'aereo
B. Pericoloso (Hazardous): failure che può avere un grande impatto negativo sulla safety o sull'efficienza delle prestazioni
C. Maggiore (Major): failure significativo, ma meno grave di A o B
D. Minore (Minor): failure evidente, ma con un impatto minore rispetto a C
E. Nessun effetto: failure che non ha alcun impatto sulla safety

Se il sistema software è classificato di livello A, deve essere testato con una copertura MC/DC del 100%. Se è di livello B, deve essere testato al 100% di copertura delle decisioni e MC/DC è opzionale. Il livello C richiede una copertura minima del 100% delle istruzioni.

Allo stesso modo, [IEC61508] è uno standard internazionale per la sicurezza funzionale di sistemi programmabili, elettronici e legati alla sicurezza. Questo standard è stato adattato per molti diversi settori tra cui l'automobilistico, ferroviario, manifatturiero, nucleare e meccanico. Il livello di criticità viene definito utilizzando una scala di livelli di integrità della safety (SIL = Safety Integrity Level), in cui SIL1 è il meno critico e SIL4 il più critico. Lo standard fornisce raccomandazioni per la copertura dei test, come mostrato nella tabella seguente (si noti che le definizioni esatte per ciascun SIL e per il significato di "raccomandato" e "altamente raccomandato" sono definite nello standard).

SIL	Copertura 100% delle istruzioni	Copertura 100% delle decisioni (rami)	Copertura 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**

Nei sistemi moderni, è raro che tutta l'elaborazione venga eseguita su un unico sistema. Il testing delle API deve essere messo in atto ogni volta che si prevede che parte dell'elaborazione sia eseguita in remoto. L'impegno da investire nel testing delle API deve essere determinato dalla criticità del sistema.

3. Tecniche analitiche - 210 minuti

Parole chiave

analisi del flusso di controllo, complessità ciclomatica, analisi del flusso dei dati, associazione definizione-utilizzo, analisi dinamica, memory leak, testing di integrazione a coppie, testing di integrazione di vicinanza, analisi statica, puntatore errato

Obiettivi di apprendimento per le tecniche analitiche

3.2 Analisi statica

TTA-3.2.1 (K3) Utilizzare l'analisi del flusso di controllo per rilevare se il codice presenta anomalie in questo flusso
TTA-3.2.2 (K2) Spiegare come viene utilizzata l'analisi del flusso di dati per rilevare se il codice presenta anomalie in questo flusso
TTA-3.2.3 (K3) Proporre modi per migliorare la manutenibilità del codice applicando l'analisi statica
TTA-3.2.4 (K2) Spiegare l'uso del grafo delle chiamate per definire strategie di test di integrazione

3.3 Analisi dinamica

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

3.1 Introduzione

Esistono due tipi di analisi: statica e dinamica.

L'analisi statica (Sezione 3.2) comprende il testing analitico che può essere svolto senza eseguire il software. Poiché il software non è in esecuzione, esso viene esaminato da uno strumento o da una persona per determinare se verrà elaborato correttamente quando verrà eseguito. Questa vista statica del software consente un'analisi dettagliata senza dover creare i dati e le condizioni preliminari che determinerebbero uno scenario in esecuzione.

Si noti che le diverse forme di review rilevanti per il Technical Test Analyst sono trattate nel Capitolo 5.

L'analisi dinamica (Sezione 3.3) richiede l'esecuzione effettiva del codice e viene utilizzata per trovare guasti che vengono rilevati più facilmente durante l'esecuzione (ad esempio, memory leak). L'analisi dinamica, come l'analisi statica, può fare affidamento su strumenti o su un individuo che monitora il sistema in esecuzione osservando indicatori quali un rapido aumento dell'uso della memoria.

3.2 Analisi statica

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

3.2.1 Analisi del flusso di controllo

L'analisi del flusso di controllo è la tecnica statica con cui vengono analizzati i percorsi che segue un programma, tramite l'uso di un grafo del flusso di controllo o di uno strumento. Parecchie anomalie possono essere rilevate in un sistema che utilizza questa tecnica, inclusi cicli mal progettati (ad esempio, con più punti di ingresso), chiamate di funzione con target ambigui in alcuni linguaggi (ad esempio, Scheme), sequenza errata delle operazioni, eccetera.

L'analisi del flusso di controllo può essere utilizzata per determinare la complessità ciclomatica. Il valore di complessità ciclomatica è un numero intero positivo che rappresenta il numero di cammini indipendenti in un grafo fortemente connesso. I cicli e le iterazioni vengono ignorati non appena sono attraversati una volta. Ogni percorso, dall'ingresso all'uscita, rappresenta un cammino univoco attraverso il modulo, e ognuno di questi andrebbe testato.

Il valore di complessità ciclomatica viene generalmente utilizzato per comprendere la complessità complessiva di un modulo. La teoria di Thomas McCabe [McCabe 76] sostiene che più complesso è il sistema, più difficile è manutenerlo e più difetti contiene. Nel corso degli anni, molti studi hanno rilevato questo tipo di correlazione tra complessità e numero di difetti contenuti. Il NIST (National Institute of Standards and Technology) raccomanda un valore massimo di complessità di 10. Qualsiasi modulo che abbia una complessità maggiore dovrebbe essere oggetto di review, per valutare una possibile suddivisione in più moduli.

3.2.2 Analisi del flusso dei dati

L'analisi del flusso dei dati copre una varietà di tecniche che riguardano la raccolta di informazioni sull'uso delle variabili di un sistema. Si esamina il ciclo di vita di ciascuna variabile (ovvero dove viene dichiarata, definita, letta, valutata e rilasciata), poiché possono verificarsi anomalie durante una qualsiasi di queste operazioni o se le operazioni sono fuori sequenza.

Una tecnica comune è la cosiddetta notazione di definizione-utilizzo, in cui il ciclo di vita di ciascuna variabile è suddiviso in tre diverse azioni atomiche:

- d: quando la variabile è dichiarata, definita o inizializzata
- u: quando la variabile viene utilizzata o letta in un calcolo o in un predicato decisionale
- k: quando la variabile viene rilasciata o esce dall'ambito (visibilità)

Una notazione alternativa comune per d-u-k è: d (define) - r (reference or read) - u (undefine).

Queste tre azioni atomiche sono combinate a coppie ("associazione definizione-utilizzo") per illustrare il flusso dei dati. Ad esempio, un "du-path" rappresenta un frammento di codice in cui viene definita e successivamente utilizzata una variabile.

Tra le possibili anomalie del flusso di dati vi sono l'esecuzione dell'azione corretta su una variabile nel momento sbagliato o l'esecuzione di un'azione errata sui dati di una variabile. Queste anomalie includono:

- Mancata assegnazione di un valore a una variabile prima di utilizzarla
- Cammino errato a causa di un valore errato in un predicato di controllo
- Tentativo di utilizzare una variabile dopo che è stata eliminata
- Riferimento a una variabile quando è fuori ambito
- Dichiarazione e rilascio di una variabile senza farne alcun utilizzo
- Ridefinizione di una variabile prima del suo utilizzo
- Mancato rilascio di una variabile allocata dinamicamente (causando un possibile memory leak)
- Modifica di una variabile che causa effetti collaterali imprevisti (ad esempio, effetti a catena quando si modifica una variabile globale senza considerare tutti gli usi della variabile)

Il linguaggio di sviluppo può guidare le regole utilizzate nell'analisi del flusso dati. I linguaggi di programmazione possono consentire al programmatore di eseguire determinate azioni con le variabili che, pur essendo consentite, possono far sì che il sistema in determinate circostanze si comporti in modo diverso rispetto a quanto previsto dal programmatore. Ad esempio, una variabile potrebbe essere definita due volte senza essere effettivamente utilizzata quando viene seguito un determinato percorso. L'analisi del flusso dati spesso etichetterà questi usi come "sospetti". Sebbene questo possa essere un uso consentito per l’assegnazione delle variabili, potrebbe portare a futuri problemi di manutenibilità nel codice.

Il test del flusso dati “utilizza il grafico del flusso di controllo per esplorare le cose irragionevoli che possono accadere ai dati "[Beizer90] e quindi trova difetti diversi rispetto all'analisi del flusso di controllo. Un Technical Test Analyst dovrebbe prendere questa tecnica in considerazione quando pianifica i test, poiché molti di questi difetti causano problemi intermittenti difficili da trovare durante l'esecuzione di test dinamici.

L'analisi del flusso dati è una tecnica statica e potrebbe non rilevare alcuni problemi che si verificano quando i dati vengono utilizzati in un sistema in esecuzione. Ad esempio, una variabile statica può contenere un puntatore ad un array creato dinamicamente che non esiste fino al momento dell’esecuzione. L'utilizzo di ambienti multiprocessore e di pre-emptive multi-tasking può creare race condition che non possono essere rilevate dall’analisi del flusso dati o del flusso di controllo.

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. Potrebbe richiedere agli sviluppatori uno sforzo maggiore per individuare e analizzare i difetti, e allo stesso tempo la modifica del codice per correggere un difetto o aggiungere una nuova funzionalità potrebbe comportare l'introduzione di ulteriori difetti.

L'analisi statica è supportata da strumenti che consentono di migliorare la manutenibilità del codice, verificando la conformità agli standard e alle linee guida di codifica. Questi standard e linee guida descrivono pratiche di codifica richieste quali naming convention, uso di commenti, indentazione e modularizzazione del codice. Si noti che gli strumenti di analisi statica generalmente producono segnalazioni piuttosto che rilevare errori. Queste segnalazioni possono essere evidenziate anche se il codice è sintatticamente corretto.

È possibile applicare strumenti di analisi statica al codice utilizzato nei siti web, per verificare la possibile esposizione a vulnerabilità di sicurezza quali code injection, cookie security, cross-site scripting, resource tampering, e SQL code injection. Ulteriori dettagli al proposito possono essere trovati nella Sezione 4.3 e nel syllabus Advanced Level Security Testing [ISTQB_ ALSEC_SYL].

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

- Aiutano a trovare codice ripetuto. Queste sezioni di codice possono essere candidate per un refactoring in moduli (sebbene il sovraccarico di runtime imposto dalle chiamate ai moduli possa essere un problema per i sistemi real-time).
- Aiutano a definire metriche che possono rivelarsi preziosi indicatori della modularizzazione del codice, quali misure di accoppiamento (coupling) e coesione (cohesion). Un sistema che deve avere una buona manutenibilità ha più probabilità di avere una bassa misura di accoppiamento (il grado con cui i moduli si basano l'uno sull'altro durante l'esecuzione) e una alta misura di coesione (il grado con cui un modulo è autonomo e focalizzato su una singola attività).
- Nel codice a oggetti, indicano dove gli oggetti derivati possono avere troppa o troppo poca visibilità nelle classi parent.
- Evidenziano aree nel codice o nell'architettura che hanno un alto livello di complessità strutturale.

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

- Attività di test più difficili
- Aumento del carico di lavoro di manutenzione
- Navigazione difficile per l'utente

3.2.4 Grafi delle chiamate

I grafi delle chiamate forniscono una rappresentazione statica della complessità della comunicazione. Sono grafi diretti in cui i nodi rappresentano i moduli del programma e gli archi rappresentano la comunicazione tra detti moduli.

I grafi delle chiamate possono essere utilizzati nel testing a livello di unità in cui funzioni o metodi diversi si chiamano a vicenda, nel testing di integrazione e di sistema quando moduli separati si chiamano l'un l'altro, o nel testing di integrazione di sistema quando sistemi separati si chiamano l'un l'altro.

I grafi delle chiamate possono essere utilizzati per i seguenti scopi:

- Progettare test che richiamano un modulo o un sistema specifico
- Determinare il numero di punti all'interno del software da cui viene chiamato un modulo o un sistema
- Valutare la struttura del codice e dell'architettura del sistema
- Fornire suggerimenti per l'ordine di integrazione (ad esempio, integrazione a coppie o di vicinanza, come discusso nel seguito)

Nel syllabus Foundation Level [ISTQB_FL_SYL], sono presentate due diverse categorie di test di integrazione: incrementale (top-down, bottom-up, ecc.) e non incrementale (big bang). Spesso viene

sostenuto che i metodi incrementali siano preferibili perché producono il codice in incrementi, rendendo così più facile l'isolamento dei difetti per via della limitata quantità di codice coinvolta.

In questo syllabus Advanced si introducono tre metodi non incrementali che utilizzano i grafi delle chiamate. Questi tre metodi possono essere preferibili ai metodi incrementali, in quanto questi ultimi possono richiedere build aggiuntive per il completamento dei test e codice accessorio per il supporto dei test. I tre metodi sono:

- Il testing di integrazione a coppie (da non confondere con la tecnica di test black-box "pairwise testing"), che è dedicato a coppie di componenti che lavorano insieme come rappresentato visivamente nel grafo delle chiamate per il testing di integrazione. Sebbene questo metodo riduca il numero di build solo di una piccola quantità, riduce la quantità necessaria di codice di test harness.
- Il testing di integrazione di vicinanza, che verifica tutti i nodi che si connettono a un dato nodo come base per il testing di integrazione. Tutti i nodi predecessori e successori di un nodo specifico nel grafo delle chiamate sono la base per i test.
- L'approccio del predicato di progettazione di McCabe, che utilizza la teoria della complessità ciclomatica applicata a un grafo di chiamata dei moduli. Ciò richiede la costruzione di un grafo delle chiamate che mostri i diversi modi in cui i moduli possono chiamarsi a vicenda, tra cui:
  - Chiamata incondizionata: la chiamata da un modulo a un altro avviene sempre
  - Chiamata condizionale: la chiamata da un modulo a un altro può avvenire a volte
  - Chiamata condizionale mutuamente esclusiva: un modulo può chiamare un e un solo modulo tra un dato numero di moduli diversi
  - Chiamata iterativa: un modulo chiama un altro almeno una volta, ma può chiamarlo più volte
  - Chiamata condizionale iterativa: un modulo può chiamarne un altro da zero a molte volte

  Dopo aver creato il grafo delle chiamate si calcola la complessità dell'integrazione e vengono creati i test per coprire il grafico.

Per ulteriori informazioni sull'uso dei grafici delle chiamate e sul testing di integrazione di vicinanza fare riferimento a [Jorgensen07].

3.3 Analisi dinamica

3.3.1 Overview

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

I failure che non sono immediatamente riproducibili (intermittenti) possono avere conseguenze significative sull'effort di testing e sulla possibilità di rilasciare o utilizzare in modo produttivo il software. Tali failure possono essere causati da memory e resource leak, da un uso non corretto dei puntatori o da altri danneggiamenti delle strutture interne (ad esempio, dello stack di sistema) [Kaner02]. A causa della natura di questi failure, che possono includere il graduale peggioramento delle prestazioni del sistema o addirittura arresti anomali, le strategie di testing devono considerarne i rischi associati e, se del caso, eseguire analisi dinamiche per ridurli (tipicamente utilizzando degli strumenti). Poiché questi failure sono spesso i più costosi da individuare e correggere, si consiglia di dare avvio all'analisi dinamica all'inizio del progetto.

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

- Prevenire il verificarsi di failure rilevando memory leak (vedere la sezione 3.3.2) e puntatori errati (vedere la sezione 3.3.3)
- Analizzare failure di sistema che non possono essere facilmente riprodotti
- Valutare il comportamento della rete
- Migliorare le prestazioni del sistema fornendo informazioni sul suo comportamento a runtime utilizzandolo per apportare modifiche ragionate

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

- Definire gli obiettivi dell'analisi dinamica
- Determinare il momento più adatto per avviare e interrompere l'analisi
- Analizzare i risultati

Durante il testing di sistema, è possibile utilizzare strumenti di analisi dinamica anche se i Technical Test Analyst hanno competenze tecniche minime; gli strumenti utilizzati solitamente creano log completi che possono essere analizzati da chi possiede le competenze tecniche necessarie.

3.3.2 Rilevamento di memory leak

I memory leak si verificano quando le aree di memoria (RAM) disponibili per un programma vengono allocate da tale programma ma non vengono successivamente rilasciate quando non più necessarie. Queste aree di memoria rimangono allocate e non sono disponibili per un eventuale riutilizzo. Quando ciò si verifica frequentemente o in situazioni di memoria insufficiente, il programma potrebbe esaurire la memoria utilizzabile. Storicamente, la gestione della memoria era responsabilità del programmatore. Qualsiasi area di memoria allocata dinamicamente doveva essere rilasciata dal programma nell'ambito di visibilità corretto 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. Identificare i memory leak nei casi in cui la memoria allocata viene liberata dalla Garbage Collection automatica può essere molto difficile.

I memory leak causano problemi che si sviluppano nel tempo e potrebbero non essere immediatamente evidenti. Questo può accadere se, ad esempio, il software è stato installato di recente o il sistema è stato riavviato, cosa che spesso si verifica durante il testing. Per questi motivi, gli effetti negativi dei memory leak possono essere notati per la prima volta quando il programma è in produzione.

Il sintomo principale di un memory leak è un costante peggioramento del tempo di risposta del sistema, che alla fine può provocare un failure del sistema. Sebbene tali failure possano essere risolti riavviando il sistema, ciò potrebbe non essere sempre pratico o possibile.

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 semplici monitor di memoria per ottenere un'indicazione generale del fatto che la memoria disponibile stia diminuendo nel tempo, sebbene sia comunque necessaria un'analisi successiva per determinare la causa esatta della diminuzione.

Ci sono anche altri tipi di leak che dovrebbero essere considerati, ad esempio di file handle, semafori e connection pool di risorse.

3.3.3 Rilevamento di puntatori errati

I puntatori errati sono puntatori che non sono più accurati e che non devono più essere utilizzati. Ad esempio, un puntatore errato potrebbe aver "perso" l'oggetto o la funzione a cui dovrebbe puntare o non puntare all'area di memoria prevista (ad esempio, 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 funziona come previsto. Questo può essere il caso in cui il puntatore errato accede a memoria non è utilizzata e teoricamente "libera" e/o contiene un valore ragionevole.
- Il programma si blocca. In questo caso il puntatore errato potrebbe aver causato un uso errato di una parte della memoria che è fondamentale per l'esecuzione del programma (ad esempio, il sistema operativo).
- Il programma non funziona correttamente perché non è possibile accedere agli oggetti richiesti. In queste condizioni il programma può continuare a funzionare, magari con un messaggio di errore.
- I dati potrebbero essere danneggiati dal puntatore e successivamente si potrebbe avere un utilizzo di valori errati (cosa che può anche rappresentare anche un problema di sicurezza)

Va notato che qualsiasi modifica apportata all'utilizzo della memoria del programma (ad esempio, una nuova build a seguito di una modifica del software) può innescare una delle quattro conseguenze sopra elencate. Ciò è particolarmente critico quando inizialmente il programma funziona come previsto nonostante l'uso di puntatori errati, e poi si blocca in modo imprevisto (in produzione o meno) a seguito di una modifica del software. È importante notare che questi problemi sono spesso sintomi di un difetto sottostante (cioè, il puntatore errato) (Fare riferimento a [Kaner02], "Lezione 74"). Vi sono strumenti che possono aiutare a identificare i puntatori errati, indipendentemente dal loro impatto sull'esecuzione del programma. Alcuni sistemi operativi dispongono di funzioni integrate per verificare in 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 si trova al di fuori dell'area di memoria consentita.

3.3.4 Analisi di efficienza prestazionale

L'analisi dinamica non è utile solo per rilevare i failure. Con l'analisi dinamica delle prestazioni, gli strumenti aiutano a identificare colli di bottiglia per l'efficienza prestazionale e a generare un'ampia gamma di metriche 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 modulo viene richiamato durante l'esecuzione. I moduli che vengono richiamati spesso sono i probabili candidati per attività di miglioramento delle prestazioni.

Unendo le informazioni sul comportamento dinamico del software alle informazioni ottenute dai grafi delle chiamate durante l'analisi statica (vedere la sezione 3.2.4), il tester può anche identificare i moduli che potrebbero essere candidati a test più dettagliati ed estesi (ad esempio, i moduli che sono richiamati frequentemente e hanno molte interfacce).

L'analisi dinamica delle prestazioni viene spesso eseguita durante i test di sistema, ma può anche essere eseguita durante il testing di un singolo sottosistema nelle fasi precedenti, utilizzando test harness. Ulteriori dettagli al proposito sono forniti nel syllabus Foundation Level Performance Testing [ISTQB_FLPT_SYL].

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

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

Obiettivi di apprendimento per le caratteristiche di qualità per il Testing Tecnico

4.2 Aspetti generali di pianificazione
TTA-4.2.1 (K4) In un dato scenario, analizzare i requisiti non funzionali e scrivere le sezioni del test plan ad essi relative
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 del ciclo di vita di sviluppo di un'applicazione in cui i test non funzionali dovrebbero essere normalmente applicati
TTA-4.2.4 (K3) In un dato scenario, definire i tipi di difetti che ci si aspetterebbe di trovare utilizzando i diversi tipi di test non funzionali

4.3 Testing di sicurezza
TTA-4.3.1 (K2) Spiegare perché è utile 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 definizione delle specifiche dei test di sicurezza

4.4 Testing di affidabilità
TTA-4.4.1 (K2) Spiegare perché è utile 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 definizione delle specifiche dei test di affidabilità

4.5 Testing di efficienza delle prestazioni
TTA-4.5.1 (K2) Spiegare perché è utile includere il testing di efficienza delle prestazioni in un approccio di test
TTA-4.5.2 (K2) Spiegare gli aspetti principali da prendere in considerazione nella pianificazione e nella definizione delle specifiche dei test di efficienza delle prestazioni

4.6 Testing di manutenibilità
TTA-4.6.1 (K2) Spiegare perché è utile includere il testing di manutenibilità in un approccio di test

4.7 Testing di portabilità
TTA-4.7.1 (K2) Spiegare perché è utile includere il testing di portabilità in un approccio di test

4.8 Testing di compatibilità
TTA-4.8.1 (K2) Spiegare perché è utile includere i test di compatibilità in un approccio di test

4.1 Introduzione

In generale, il Technical Test Analyst concentra i test sul "come" funziona il prodotto, piuttosto che sugli aspetti funzionali di "cosa" fa. I test possono aver luogo a qualsiasi livello di test. Ad esempio, durante il testing delle componenti di sistemi embedded 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 di sistema, è utile testare aspetti di affidabilità quali ad es. 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 vari server, client, database, reti e altre risorse. Indipendentemente dal livello di test, il testing deve essere condotto in base alle priorità di rischio e alle risorse disponibili.

Va notato che per testare le caratteristiche di qualità non funzionali descritte in questo capitolo possono essere applicati sia il testing dinamico che il testing statico (vedere il Capitolo 3).

La descrizione delle caratteristiche e sotto-caratteristiche di qualità del prodotto fa riferimento allo standard ISO 25010 [ISO25010]. La tabella sottostante riporta tali aspetti, insieme all'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, riusabilità		X
Portabilità	Adattabilità, installabilità, sostituibilità		X
Sicurezza	Confidenzialità, integrità, non-repudiation, responsabilità, autenticità		X
Compatibilità	Co-esistenza		X
	Interoperabilità	X	

L'appendice A riporta una tabella che confronta le caratteristiche descritte nello standard ISO 9126 (utilizzato nella versione 2012 di questo syllabus) con quelle nel più recente ISO 25010.

Per tutte le caratteristiche e le sotto-caratteristiche discusse in questa sezione, occorre riconoscere i rischi tipici in modo che possa essere approntato e documentato un adeguato approccio di test. I test delle caratteristiche di qualità richiedono un'attenzione particolare alla tempistica del ciclo di vita, agli strumenti, agli standard, alla disponibilità di software e documentazione, nonché all'esperienza tecnica. Senza la pianificazione di un approccio dedicato ad ogni caratteristica che consenta di tenere conto delle sue specifiche esigenze di testing, il tester potrebbe non riuscire a inserire nel proprio programma tempi adatti di pianificazione, preparazione ed esecuzione [Bath14].

Alcuni di questi tipi di test, ad esempio i test di efficienza delle prestazioni, richiedono una pianificazione approfondita, attrezzature dedicate, strumenti specifici, capacità di testing specializzate e, nella maggior parte dei casi, una notevole quantità di tempo. Il testing delle caratteristiche di qualità e delle sotto-caratteristiche deve essere integrato nel programma complessivo dei test con adeguate risorse.

Ciascuno di questi tipi di test ha esigenze specifiche, si rivolge a problemi specifici e può verificarsi in momenti diversi durante il ciclo di vita dello sviluppo del software, come discusso nelle sezioni che seguono.

La compilazione e il reporting delle metriche relative alle caratteristiche di qualità e alle sotto-caratteristiche è un compito affidato al Test Manager, mentre il Test Analyst o il Technical Test Analyst (in base alla tabella soprariportata) hanno il compito di raccogliere le informazioni per ciascuna metrica.

Le metriche di qualità raccolte nei test di pre-produzione dal Technical Test Analyst possono costituire la base dei livelli di servizio (SLA) tra il fornitore e gli stakeholder del sistema software (ad esempio, clienti, operatori). In alcuni casi, i test continuano a essere eseguiti dopo che il software è entrato in produzione, spesso da un team o da un'organizzazione separati, ad esempio per l'efficienza delle prestazioni e per l'affidabilità, in quanto possono mostrare risultati diversi nell'ambiente di produzione rispetto all'ambiente di test.

4.2 Aspetti generali di pianificazione

La mancata pianificazione di test non funzionali può mettere a rischio il successo di un'applicazione. Il Test Manager può quindi chiedere al Technical Test Analyst di identificare i principali rischi per le caratteristiche di qualità più rilevanti (vedi tabella nella Sezione 4.1) e di affrontare eventuali problemi di pianificazione associati ai test proposti. Queste informazioni possono poi essere utilizzate per creare il master test plan.

Nell'affrontare questo tema vi sono alcuni fattori generali da prendere in considerazione:

- I requisiti degli stakeholder
- La necessità di strumenti e di formazione
- Requisiti dell'ambiente di test
- Considerazioni organizzative
- Considerazioni sulla sicurezza dei dati
- Rischi e difetti tipici

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 raccogliere dagli stakeholder le aspettative riguardanti le caratteristiche di qualità tecnica e valutare i rischi che queste rappresentano.

Un approccio di comune utilizzo è di dare per assunto 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, cosa che 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 potrebbero avere difficoltà a specificare i loro requisiti.

Quando si acquisiscono requisiti non funzionali è consigliabile raccogliere più punti di vista da parte degli stakeholder, ad esempio clienti, product owner, utenti, personale operativo e personale addetto alla manutenzione. Il mancato coinvolgimento degli stakeholder aumenta la probabilità che alcuni requisiti non vengano rispettati. Per maggiori dettagli sull'acquisizione dei requisiti, fare riferimento anche al syllabus Advanced Test Manager [ISTQB_ALTM_SYL].

Nei progetti Agile i requisiti non funzionali possono essere descritti come user story o essere aggiunti come vincoli non funzionali a una funzionalità definita nei casi d'uso.

4.2.2 Necessità di strumenti e formazione

Strumenti o simulatori sono particolarmente importanti per il test dell'efficienza delle prestazioni e alcuni test di sicurezza. I Technical Test Analyst hanno il compito di stimare i costi e i tempi necessari per acquisire, apprendere e implementare tali strumenti. Laddove debbano essere utilizzati strumenti specializzati, la pianificazione deve tenere conto delle curve di apprendimento per nuovi strumenti e/o del costo dell'assunzione di specialisti esterni degli strumenti.

Lo sviluppo di un simulatore complesso può rappresentare un progetto di sviluppo a sé stante e va pianificato in quanto tale. In particolare, nel programma e nel piano delle risorse occorre prendere in considerazione il testing e la documentazione dello strumento sviluppato. È necessario pianificare un budget e un tempo sufficienti per aggiornare e ripetere il testing del simulatore man mano che il prodotto simulato cambia. La pianificazione dei simulatori da utilizzare in applicazioni critiche per la sicurezza deve tenere conto del testing di accettazione e dell'eventuale certificazione del simulatore da parte di un ente indipendente.

4.2.3 Requisiti dell'ambiente di test

Molti test tecnici (ad esempio, test di sicurezza, test di efficienza delle prestazioni), per fornire misure realistiche richiedono un ambiente di test simile alla produzione. A seconda delle dimensioni e della complessità del sistema sottoposto a test, questo può avere un impatto significativo sulla pianificazione e sul budget dedicato ai test. Poiché il costo di tali ambienti può essere elevato, è possibile considerare le seguenti alternative:

- Utilizzo dell'ambiente di produzione
- Utilizzo di una versione ridotta del sistema, avendo cura che i risultati dei test siano sufficientemente rappresentativi del sistema di produzione
- Utilizzo di risorse su cloud come alternativa all'acquisizione diretta delle risorse
- Utilizzo di ambienti virtualizzati

La tempistica di esecuzione di tali test deve essere pianificata attentamente in quanto è molto probabile che gli stessi possano essere eseguiti solo in momenti specifici (ad esempio, in tempi di ridotto utilizzo del sistema).

4.2.4 Considerazioni organizzative

In un sistema completo, i test tecnici possono comportare la misurazione del comportamento di diverse componenti (ad esempio, server, database, reti). Se queste componenti sono distribuite su un numero di siti e organizzazioni differenti, lo sforzo richiesto per pianificare e coordinare i test può essere significativo. Ad esempio, alcuni componenti software potrebbero essere disponibili solo in particolari momenti della giornata o dell'anno, oppure le diverse organizzazioni potrebbero dare 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" a scopo di testing (ad esempio, esperti "presi in prestito") può comportare gravi interruzioni dei test programmati.

4.2.5 Considerazioni sulla sicurezza dei dati

Nella fase di pianificazione del testing occorre prendere in considerazione le specifiche misure di sicurezza implementate per un sistema, per garantire che tutte le previste attività di testing 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 regole e le leggi sulla protezione dei dati possono precludere la generazione di dati di test che si basino sui dati di produzione (ad esempio, dati personali, dati della carta di credito). Rendere anonimi i dati del test è un'attività non banale, che deve essere pianificata all'interno dell'implementazione del test.

4.2.6 Rischi e difetti tipici

Identificare e gestire i rischi è fondamentale per la pianificazione del testing (vedere il Capitolo 1). Il Technical Test Analyst identifica i rischi del prodotto utilizzando la propria conoscenza dei comuni tipi di difetti attesi per una particolare caratteristica di qualità. Ciò consente di selezionare i tipi di test necessari per affrontare tali rischi. Questi aspetti specifici sono trattati nelle restanti sezioni di questo capitolo che descrivono le singole caratteristiche di qualità.

4.3 Testing di sicurezza

4.3.1 Ragioni per prendere in considerazione il testing di sicurezza

Il testing di sicurezza valuta la vulnerabilità di un sistema tentando di compromettere la politica di sicurezza del sistema stesso. Di seguito è riportato un elenco 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 esempio, capacità di eseguire attività per le quali l'utente non dispone dei diritti). Questi test ruotano intorno ai diritti e ai privilegi di accesso degli utenti, informazioni che dovrebbero essere disponibili nelle specifiche.
- Software che manifesta effetti collaterali indesiderati durante l'esecuzione della funzione prevista. Ad esempio, un lettore multimediale che riproduce correttamente l'audio ma lo fa scrivendo file in una memoria temporanea non crittografata, manifesta un effetto collaterale che può essere sfruttato dai pirati del software.
- Codice inserito in una pagina web che può essere utilizzato da utenti successivi (cross-site scripting o XSS). Questo codice potrebbe essere dannoso.
- Buffer overflow (buffer overrun) che può essere causato dall'immissione in un campo di input dell'interfaccia utente di stringhe più lunghe di quanto il codice possa gestire correttamente. Una vulnerabilità di buffer overflow rappresenta un'opportunità per eseguire 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 esempio, transazioni con carta di credito) da parte di terzi in cui un utente non venga a conoscenza della presenza di tale terza parte (attacco "Man in the Middle")
- Violazione dei codici di crittografia utilizzati per proteggere i dati sensibili.
- Bombe logiche (a volte dette Uova di Pasqua – Easter Egg), che possono essere inserite intenzionalmente nel codice e che si attivano solo in determinate condizioni (ad esempio, in una data specifica). Quando le bombe logiche si attivano, possono eseguire azioni dannose quali l'eliminazione di file o la formattazione dei dischi.

4.3.2 Pianificazione dei test di sicurezza

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

- Poiché è possibile introdurre problemi di sicurezza durante la progettazione architetturale e l'implementazione del sistema, è possibile programmare il testing di sicurezza per i livelli di unit test, test di integrazione e test di sistema. A causa della natura mutevole delle minacce alla sicurezza, i test di sicurezza possono anche essere programmati con regolarità dopo che il sistema è entrato in produzione. Ciò è particolarmente vero per architetture dinamiche aperte quali 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 dell'architettura, del progetto e del codice, nonché l'analisi statica del codice con strumenti di

sicurezza. Questi possono essere efficaci per trovare 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 (vedere di seguito) che richiedono un'attenta 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 esempio, testare i diritti e i privilegi di accesso degli utenti).
- Un aspetto essenziale della pianificazione del testing di sicurezza è ottenerne l'approvazione. Per il Technical Test Analyst ciò significa assicurarsi che sia stato ottenuto il permesso esplicito dal Test Manager 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 dimostri le intenzioni e l'autorizzazione, la scusa "stavamo eseguendo un test di sicurezza" potrebbe essere difficile da spiegare in modo convincente.
- La pianificazione del testing di sicurezza dovrebbe essere coordinata con il responsabile della sicurezza delle informazioni di un'organizzazione, se l'organizzazione ha un tale ruolo.
- Va notato che i miglioramenti che possono essere apportati alla sicurezza di un sistema possono influenzarne l'efficienza delle prestazioni o l'affidabilità. Dopo aver apportato miglioramenti alla sicurezza, è consigliabile prendere in considerazione la necessità di condurre test di efficienza delle prestazioni o di affidabilità (vedere le sezioni 4.4 e 4.5 di seguito).

Per la pianificazione dei test di sicurezza si possono applicare specifici standard quali ad esempio [ISA /IEC 62443-3-2], che si applica all'automazione industriale e ai sistemi di controllo.

Il syllabus Advanced Level Security Testing [ISTQB_ALSEC_SYL] presenta ulteriori dettagli sugli aspetti chiave della pianificazione del testing di sicurezza.

4.3.3 Specifica dei test di sicurezza

I test di sicurezza possono essere raggruppati [Whittaker04] in base all'origine del rischio di sicurezza, ad esempio:

- Interfaccia utente: accesso non autorizzato e input dannosi
- File system: accesso ai dati sensibili archiviati in file o repository
- Sistema operativo: archiviazione di informazioni sensibili quali la password in forma non crittografata, che potrebbe essere esposta quando il sistema si blocca tramite input dannosi
- Software esterno: interazioni che possono verificarsi tra componenti esterne utilizzate dal sistema, sia a livello di rete (ad esempio, pacchetti o messaggi errati che vengono trasmessi) sia a livello di componente software (ad esempio, guasto di un componente software su cui si basa il software)

Anche le sotto-caratteristiche della sicurezza in ISO 25010 [ISO25010] possono fornire una base per definire le specifiche dei test di sicurezza, in particolare sui seguenti aspetti:

- Riservatezza: il grado con cui un prodotto o un sistema garantisce che i dati siano accessibili solo a coloro che sono autorizzati ad accedervi
- Integrità: il grado con cui un sistema, prodotto o componente impedisce l'accesso non autorizzato o la modifica di programmi o dati
- Non-repudiation: il grado con cui è possibile dimostrare che certe azioni o eventi si sono verificati in modo che ciò non possa essere negato in seguito
- Responsabilità: il grado con cui le azioni di un'entità possono essere ricondotte in modo univoco all'entità stessa
- Autenticità: il grado con cui è possibile dimostrare che l'identità di un soggetto o di una risorsa è quella dichiarata

Per lo sviluppo di test di sicurezza si può usare il seguente approccio [Whittaker04]:

- Raccogliere informazioni che possono essere utili per specificare i test, ad esempio nomi di dipendenti, indirizzi fisici, dettagli riguardanti le reti interne, numeri IP, identità del software o hardware utilizzato e versione del sistema operativo.
- Eseguire una scansione delle vulnerabilità utilizzando strumenti disponibili. Tali strumenti non vengono utilizzati direttamente per compromettere il/i sistema/i, ma per identificare le vulnerabilità che sono o che possono provocare una violazione della politica di sicurezza. Vulnerabilità specifiche possono anche essere identificate utilizzando informazioni e checklist quali quelle fornite dal National Institute of Standards and Technology (NIST) [Web-1] e dall'Open Web Application Security Project ™ (OWASP) [Web-4].
- Sviluppare "piani di attacco" (ovvero 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 tramite varie interfacce (ad esempio, interfaccia utente, file system). I vari "attacchi" descritti in [Whittaker04] sono una preziosa fonte di tecniche sviluppate specificamente per i test di sicurezza.

Si noti che è possibile sviluppare piani di attacco anche per i test di penetrazione (vedere [ISTQB_ALSEC_SYL]).

I problemi di sicurezza possono essere identificati tramite review (vedere il Capitolo 5) e/o con l'uso di strumenti di analisi statica (vedere la Sezione 3.2). Gli strumenti di analisi statica contengono un ampio insieme di regole specifiche per controllare il codice rispetto a minacce alla sicurezza, ad esempio per controllare problemi di buffer overflow causati dal mancato controllo della dimensione del buffer prima dell'assegnazione dei dati.

Dettagli aggiuntivi sui test di sicurezza si possono trovare nella Sezione 3.2 (analisi statica) e nel syllabus Advanced Level Security Testing [ISTQB_ALSEC_SYL].

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: il grado con cui una componente o un sistema soddisfa le esigenze di affidabilità durante il normale funzionamento
- Tolleranza ai guasti: la capacità del prodotto software di mantenere un livello specifico di prestazione in caso di difetti del software o di violazione della sua interfaccia specificata
- Recuperabilità: la capacità del prodotto software di ristabilire un livello specifico di prestazione e recuperare i dati direttamente interessati in caso di guasto
- Disponibilità: il grado con cui una componente o un sistema risulta operativo e accessibile quando ne è richiesto l'uso

4.4.2 Misurazione della maturità del software

Un obiettivo del testing di affidabilità è di monitorare nel tempo una misura statistica della maturità del software e confrontarla con un obiettivo desiderato di affidabilità, espresso come livello di servizio (SLA). Le misure possono prendere la forma di Mean Time Between Failures (MTBF), Mean Time To Repair (MTTR), o qualsiasi altra forma di misurazione dell'intensità del guasto (ad esempio, il numero di guasti di una particolare gravità che si verificano in una settimana). Queste misure possono essere utilizzate come criteri di uscita (ad esempio, per il rilascio di produzione).

Si prega di notare che il termine “maturità” nel contesto dell’affidabilità non deve essere confuso con la maturità dell'intero processo di test del software che viene discusso nel syllabus ISTQB Advanced Level Test Manager [ISTQB_ALTM_SYL].

4.4.3 Testing di tolleranza ai guasti

In aggiunta ai test funzionali che valutano la tolleranza del software ai guasti in termini di gestione di valori di input imprevisti (i cosiddetti test negativi), occorre inserire anche test per valutare la tolleranza di un sistema ai guasti che si verificano esternamente all'applicazione sottoposta a test. Tali guasti vengono generalmente segnalati dal sistema operativo (ad esempio, disco pieno, processo o servizio non disponibile, file non trovato, memoria non disponibile). I test di tolleranza ai guasti a livello di sistema possono essere supportati da strumenti specifici.

Si noti che quando si parla di tolleranza ai guasti si usano comunemente anche i termini "robustezza" e "tolleranza agli errori" (vedere per dettagli [ISTQB_GLOSSARY]).

4.4.4 Testing di recuperabilità

Il testing di affidabilità include anche altre forme di test per valutare la capacità del sistema di recuperare da guasti hardware o software in un modo predeterminato che consenta successivamente di ripristinare la normale operatività. I test di recuperabilità includono i test di failover e di backup e restore.

I test di failover vengono eseguiti quando le conseguenze di un guasto sono così negative che il sistema implementa specifiche misure hardware e/o software per garantire il funzionamento anche in caso di guasto. I test di failover possono essere applicabili, ad esempio, dove il rischio di perdite finanziarie è estremo o dove esistono problemi critici per la safety. Laddove i failure possono derivare da eventi catastrofici, questa forma di testing di recuperabilità può anche essere chiamata testing di "recupero di emergenza" (“disaster recovery”).

Le tipiche misure preventive per i guasti hardware includono il bilanciamento del carico tra diversi processori e server, processori o dischi in cluster, in modo che un elemento possa immediatamente subentrare a un altro in caso di guasto (sistemi ridondanti). Una tipica misura per il software potrebbe essere l'implementazione, nei cosiddetti sistemi dissimili ridondanti, di più di un'istanza indipendente di un sistema (ad esempio, il sistema di controllo del volo di un aereo). I sistemi ridondanti sono tipicamente una combinazione di misure software e hardware e possono essere chiamati sistemi duplex, triplex o quadruplex, a seconda del numero di istanze indipendenti (rispettivamente due, tre o quattro). La dissimilarità si ottiene quando gli stessi requisiti software sono forniti a due (o più) team di sviluppo indipendenti e non collegati, con l'obiettivo di avere gli stessi servizi forniti con software diverso. Ciò protegge i sistemi dissimili ridondanti in quanto è meno probabile che un input difettoso simile produca lo stesso risultato. Queste misure adottate per migliorare la recuperabilità di un sistema possono influenzare direttamente anche la sua affidabilità, quindi dovrebbero essere prese in considerazione anche nel testing di affidabilità.

Il test di failover è progettato per testare in modo esplicito i sistemi simulando le modalità di failure o causando effettivamente failure in un ambiente controllato. A seguito di un failure, il meccanismo di failover viene testato per garantire che i dati non vadano persi o danneggiati e che i livelli di servizio concordati vengano mantenuti (ad esempio, disponibilità delle funzioni o tempi di risposta).

I test di backup e restore si concentrano sulle misure procedurali impostate per ridurre al minimo gli effetti di un failure. Tali test valutano le procedure (solitamente documentate in un manuale) per effettuare il backup in varie modalità ed effettuare il restore dei dati in caso di perdita o danneggiamento degli stessi. I test case sono progettati per garantire che i percorsi critici di ogni procedura siano coperti. È possibile eseguire review tecniche per il "dry-run" questi scenari e convalidare i manuali rispetto alle

procedure effettive. Il testing di accettazione operativa valuta gli scenari in un ambiente di produzione o simile alla produzione per convalidare il loro utilizzo effettivo.

Le misure relative ai test di backup e restore possono essere:

- Il tempo impiegato per eseguire diversi tipi di backup (ad esempio, completo, incrementale)
- Il tempo impiegato per eseguire il restore dei dati
- I livelli garantiti di backup dei dati (ad esempio, ripristino di tutti i dati non più vecchi di 24 ore, ripristino di dati di transazioni specifici non più vecchi di un'ora)

4.4.5 Testing di disponibilità

Qualunque sistema che abbia interfacce con altri sistemi e/o processi (ad esempio, per ricevere input) fa affidamento sulla disponibilità di tali interfacce per garantire l'operatività complessiva.

Gli obiettivi principali del test di disponibilità sono:

- Determinare se le componenti e i processi di sistema sono disponibili (su richiesta o continuamente) e sono in grado di rispondere come previsto alle richieste.
- Fornire misurazioni da cui è possibile ottenere un livello complessivo di disponibilità (spesso indicato negli SLA come percentuale di tempo di disponibilità).
- Determinare se un sistema è pronto per il funzionamento (ad esempio in qualità di criterio per i test di accettazione operativa).

Il testing di disponibilità viene eseguito sia prima che dopo l'entrata in servizio operativo ed è particolarmente rilevante per le seguenti situazioni:

- Dove i sistemi sono costituiti da altri sistemi (cioè, sistemi di sistemi). I test si concentrano sulla disponibilità di tutti i singoli sistemi.
- Quando un sistema o un servizio sono forniti dall'esterno (ad esempio, da un fornitore di terze parti). I test si concentrano sulla misurazione dei livelli di disponibilità per garantire il rispetto dei livelli di servizio concordati.

La disponibilità può essere misurata utilizzando strumenti di monitoraggio dedicati o eseguendo test specifici. Tali test sono tipicamente automatizzati e possono essere eseguiti parallelamente alla normale operatività, a condizione che non influiscano su di essa (ad esempio, riducendo l'efficienza delle prestazioni).

4.4.6 Pianificazione dei test di affidabilità

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

- L'affidabilità può continuare a essere monitorata dopo che il software è entrato in produzione. L'organizzazione e il personale responsabile del funzionamento del software devono essere consultati quando si raccolgono i requisiti di affidabilità ai fini della pianificazione dei test.
- Il Technical Test Analyst può scegliere un modello di crescita dell'affidabilità che mostri i livelli attesi di affidabilità nel tempo. Un modello di crescita dell'affidabilità può fornire informazioni utili al Test Manager attraverso il confronto dei livelli di affidabilità attesi e raggiunti.
- I test di affidabilità devono essere eseguiti in un ambiente simile alla produzione. L'ambiente utilizzato dovrebbe rimanere il più stabile possibile per consentire il monitoraggio dell'andamento dell'affidabilità nel tempo.
- Poiché i test di affidabilità spesso richiedono l'uso dell'intero sistema, il testing di affidabilità viene condotto più comunemente come parte del testing di sistema. Tuttavia, sia le singole componenti sia insiemi integrati di componenti possono essere sottoposti a test di affidabilità. L'architettura di dettaglio, la progettazione e le review del codice possono essere utilizzate anche per rimuovere alcuni dei rischi di problemi di affidabilità nell'implementazione del sistema.

- Al fine di produrre risultati di test che siano statisticamente significativi, i test di affidabilità di solito richiedono lunghi tempi di esecuzione, cosa che può rendere difficile la pianificazione di questi test all'interno di altri test pianificati.

4.4.7 Specifica dei test di affidabilità

I test di affidabilità possono assumere la forma di una serie ripetuta di test predeterminati. Questi possono essere test selezionati in modo casuale da un pool oppure test case generati da un modello statistico utilizzando metodi casuali o pseudo-casuali. I test possono anche essere basati su modelli di utilizzo a volte indicati come "Profili Operativi" (vedere la Sezione 4.5.3).

Laddove i test di affidabilità sono programmati per essere eseguiti automaticamente in parallelo alla normale operatività (ad esempio, per testare la disponibilità), sono generalmente definiti in modo da essere i più semplici possibili, per evitare possibili impatti negativi sull'efficienza delle prestazioni del sistema.

Alcuni test di affidabilità possono indicare che le azioni che richiedono un uso intensivo della memoria devono essere eseguite ripetutamente, in modo da rilevare eventuali memory leak.

4.5 Testing di efficienza delle prestazioni

4.5.1 Tipi di testing di efficienza delle prestazioni

4.5.1.1 Load Test

Il load testing si concentra sulla capacità di un sistema di gestire livelli realisticamente crescenti di carico derivanti da richieste di transazione generate da un numero di utenti o processi simultanei. È utile a misurare e analizzare i tempi di risposta medi per gli utenti in diversi scenari di utilizzo tipico (profili operativi). Vedi anche [Splaine01].

4.5.1.2 Stress Test

Lo stress testing si concentra sulla capacità di un sistema o di una componente di gestire picchi di carico pari o superiori ai limiti dei carichi di lavoro previsti o definiti nelle specifiche, oppure con una disponibilità ridotta di risorse quali la larghezza di banda disponibile. I livelli di prestazione del sistema dovrebbero degradare lentamente e in modo prevedibile senza failure all'aumentare del livello di stress. In particolare, si deve testare l'integrità funzionale del sistema mentre il sistema è sotto stress, al fine di trovare possibili difetti nell'elaborazione funzionale o incoerenze dei dati.

Uno degli obiettivi dello stress testing è di scoprire i limiti oltre i quali un sistema va in crisi, in modo da poter determinare "l'anello più debole della catena". Lo stress testing consente di aggiungere capacità al sistema in modo tempestivo (ad es. memoria, capacità della CPU, archiviazione dei dati).

4.5.1.3 Testing di scalabilità

Il testing di scalabilità si concentra sulla capacità di un sistema di soddisfare requisiti futuri di efficienza, che potrebbero essere superiori a quelli attualmente richiesti. L'obiettivo dei test è determinare la capacità del sistema di crescere (ad esempio, con più utenti, maggiori quantità di dati archiviati) senza raggiungere un punto in cui i requisiti prestazionali definiti nelle specifiche non possono essere soddisfatti o il sistema va in crisi. Una volta che i limiti di scalabilità sono noti, i valori di soglia possono essere impostati e monitorati nell'ambiente di produzione per fornire segnalazioni di problemi imminenti. Inoltre, l'ambiente di produzione può essere dimensionato con risorse hardware adeguate a soddisfare le esigenze previste.

4.5.2 Pianificazione dei test di efficienza delle prestazioni

Oltre ai problemi di pianificazione generale descritti nella Sezione 4.2, altri fattori che possono influenzare la pianificazione dei test di efficienza delle prestazioni sono:

- A seconda dell'ambiente di test utilizzato e del software da testare, (vedere la Sezione 4.2.3) i test di efficienza delle prestazioni possono necessitare dell'intero sistema per poter essere efficaci. Se così, il testing di efficienza delle prestazioni viene di solito pianificato durante il testing di sistema. Altri test di efficienza delle prestazioni che possono essere eseguiti efficacemente a livello di componente possono essere programmati durante lo unit testing.
- In generale, è raccomandabile eseguire test iniziali di efficienza delle prestazioni il prima possibile, anche se un ambiente di produzione o simile non è ancora disponibile. Questi primi test possono rilevare problemi di efficienza delle prestazioni (ad esempio, colli di bottiglia) e ridurre il rischio del progetto, evitando costose correzioni in fasi successive dello sviluppo o in produzione.
- Le review del codice, in particolare quelle incentrate sull'interazione con il database, sull'interazione tra componenti e sulla gestione degli errori, possono identificare problemi di efficienza delle prestazioni (specialmente per quanto riguarda la logica di "wait and retry" e le query inefficienti). Per questa ragione tali review dovrebbero essere pianificate all'inizio del ciclo di vita dello sviluppo del software.
- L'hardware, il software e la larghezza di banda di rete necessari per eseguire i test di efficienza delle prestazioni, vanno identificati a livello di pianificazione e ne vanno previsti i costi. Le esigenze dipendono principalmente dal carico da generare, che dipende dal numero di utenti virtuali da simulare e dalla quantità di traffico di rete che è probabile essi sia in grado di generare. Non prendere questi aspetti in considerazione potrebbe portare a misurazioni non rappresentative delle prestazioni. Ad esempio, la verifica dei requisiti di scalabilità di un sito Internet molto visitato può richiedere la simulazione di centinaia di migliaia di utenti virtuali.
- La generazione del carico richiesto per i test di efficienza delle prestazioni può avere un'influenza significativa sui costi di acquisizione di hardware e strumenti. Occorre prendere in considerazione questo aspetto nella pianificazione dei test di efficienza delle prestazioni, per garantire che siano disponibili coperture economiche adeguate.
- I costi di generazione del carico per i test di efficienza delle prestazioni possono essere ridotti al minimo noleggiando l'infrastruttura di test richiesta. Ciò può comportare, ad esempio, il noleggio di licenze "ricaricabili" (“top-up”) per strumenti prestazionali o l'utilizzo dei servizi di un fornitore terzo per soddisfare le necessità di hardware (ad es. servizi cloud). Se si adottasse questo approccio, il tempo a disposizione per lo svolgimento dei test di efficienza delle prestazioni potrebbe essere limitato e ciò va quindi pianificato con attenzione.
- È necessario prestare attenzione in fase di pianificazione al fine di assicurare che lo strumento di test delle prestazioni fornisca la compatibilità richiesta con i protocolli di comunicazione utilizzati dal sistema sotto test.
- I difetti legati all'efficienza delle prestazioni spesso hanno un impatto significativo sul sistema sotto test. Quando i requisiti di efficienza delle prestazioni sono di assoluta importanza, è spesso utile condurre test di efficienza delle prestazioni sulle componenti critiche (tramite driver e stub) in modo che i test possano iniziare all'inizio del ciclo di vita invece di attendere il test di sistema.

Dettagli aggiuntivi sulla pianificazione dei test di efficienza delle prestazioni sono disponibili nel syllabus Foundation Level Performance Testing [ISTQB_FLPT_SYL].

4.5.3 Definizione Specifica dei test di efficienza delle prestazioni

La definizione dei test per i diversi tipi di test di efficienza delle prestazioni quali il load o lo stress test si basa sulla definizione di profili operativi (operational profile), che rappresentano forme distinte di comportamento dell'utente durante l'interazione con un'applicazione. Per una data applicazione possono esserci più profili operativi.

Il numero di utenti inclusi in profilo operativo può essere ricavato tramite l'utilizzo di strumenti di monitoraggio (dove l'applicazione effettiva o una comparabile è già disponibile) o facendo previsioni sull'utilizzo del sistema. Tali previsioni possono essere basate su algoritmi o fornite dall'organizzazione e sono particolarmente importanti per definire i profili operativi da utilizzare nei test di scalabilità.

I profili operativi costituiscono la base per definire il numero e il tipo di test case da utilizzare durante i test di efficienza delle prestazioni. Questi test sono spesso controllati da strumenti di test che creano utenti "virtuali" o simulati in quantità che rappresenteranno il profilo sotto test (vedere la Sezione 6.2.2).

Dettagli aggiuntivi sulla progettazione dei test di efficienza delle prestazioni sono disponibili nel syllabus Foundation Level Performance Testing [ISTQB_FLPT_SYL].

4.5.4 Sotto-caratteristiche di qualità per l'efficienza delle prestazioni

La classificazione ISO 25010 delle caratteristiche di qualità del prodotto include le seguenti sotto-caratteristiche di efficienza delle prestazioni:

- Comportamento nel tempo: la capacità di una componente o un sistema di rispondere agli input di utente o sistemi entro un tempo definito e in condizioni definite
- Utilizzo delle risorse: la capacità del prodotto di utilizzare quantità e tipi adeguati di risorse
- Capacità: il limite massimo di gestione per un particolare parametro

4.5.4.1 Comportamento temporale

Il comportamento temporale si concentra sulla capacità di una componente o un sistema di rispondere agli input dell'utente o del sistema entro un tempo definito e in condizioni definite. Le misurazioni del comportamento temporale variano in base agli obiettivi del test. Per le singole componenti software, il comportamento temporale può essere misurato in base ai cicli della CPU, mentre per i sistemi basati su client può essere misurato in base al tempo impiegato per rispondere a una particolare richiesta dell'utente. Per i sistemi le cui architetture sono costituite da diverse componenti (ad esempio, client, server, database) le misurazioni del comportamento temporale sono associate alle transazioni tra le singole componenti, in modo da poter identificare i "colli di bottiglia".

4.5.4.2 Utilizzo delle risorse

I test relativi all'utilizzo delle risorse valutano l'utilizzo delle risorse di sistema (ad es. utilizzo della memoria, capacità del disco, larghezza di banda della rete, connessioni) rispetto a un benchmark predefinito. Questi parametri vengono confrontati sia in condizioni di carico normali che in situazioni di stress, ad esempio in caso di elevati livelli di transazioni e di dati, per determinare se si stia verificando una crescita anomala di utilizzo.

Ad esempio, per i sistemi embedded real-time, l'utilizzo della memoria (a volte indicato come "memory footprint") svolge un ruolo significativo nei test di efficienza delle prestazioni. Se il memory footprint supera la soglia consentita, il sistema potrebbe non disporre di memoria sufficiente per eseguire le proprie attività nei periodi di tempo definiti, cosa che potrebbe rallentare il sistema o addirittura causarne un arresto anomalo.

Per rilevare colli di bottiglia di efficienza delle prestazioni si può applicare anche l'analisi dinamica alla valutazione dell'utilizzo delle risorse (vedere la Sezione 3.3.4).

4.5.4.3 Capacità

La capacità di un sistema (software e hardware inclusi) rappresenta il limite massimo di gestione di un particolare parametro. I requisiti di capacità sono in genere definiti dagli stakeholder tecnici e operativi e possono riguardare parametri quali il numero massimo di utenti che possono utilizzare un'applicazione

in un dato momento, il volume massimo di dati che può essere trasmesso al secondo (cioè, la larghezza di banda) e il numero massimo di transazioni che possono essere gestite al secondo.

L'approccio di test per i limiti di capacità è generalmente simile all'approccio descritto nelle sezioni 4.5.2 e 4.5.3 per il testing di efficienza delle prestazioni. I profili operativi per i test di capacità si concentrano sulla generazione di un carico che vada a stressare un particolare limite, ad esempio la generazione di un carico che sottopone il sistema alla quantità massima di trasferimento dati. Gli approcci allo stress testing e al testing di scalabilità possono essere applicati anche alla capacità, per testare il comportamento del sistema oltre i limiti di capacità specificati (vedere rispettivamente le sezioni 4.5.1.2 e 4.5.1.3).

4.6 Testing di manutenibilità

Nella vita del software, spesso è molto più lungo il tempo di manutenzione che quello di sviluppo. Il testing di manutenzione viene svolto per verificare l'impatto delle modifiche su un sistema in esercizio operativo o sul corrispondente ambiente di produzione. Per garantire che la manutenzione sia la più efficiente possibile, può essere svolto del testing di manutenibilità volto a misurare la facilità con cui il codice può essere analizzato, modificato e testato.

Gli obiettivi tipici di manutenibilità per gli stakeholder, quali l'owner del software o il personale operativo, includono:

- La minimizzazione del costo di ownership o del costo di esercizio operativo
- La minimizzazione dei downtime per la manutenzione

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

- Elevata probabilità di modifiche al software dopo la messa in produzione (ad esempio, per correggere difetti o introdurre aggiornamenti pianificati)
- Gli stakeholder interessati ritengono che i vantaggi derivanti dal raggiungimento degli obiettivi di manutenibilità durante lo sviluppo superino i costi di esecuzione dei test di manutenibilità e di gestione delle richieste di modifica
- Rischi di una scarsa manutenibilità del software (ad es. lunghi tempi di risposta per i difetti segnalati dagli utenti e/o dai clienti) giustificano l'esecuzione di test di manutenibilità

4.6.1 Testing di manutenibilità statico e dinamico

Le tecniche più adatte per il testing di manutenibilità includono le analisi statiche e le review, come discusso nelle sezioni 3.2 e 5.2. Il testing di manutenibilità deve essere avviato non appena la documentazione di progetto è disponibile e dovrebbe continuare durante tutto lo sviluppo. Poiché la manutenibilità è incorporata nel codice e nella documentazione per ogni singola componente di codice, la manutenibilità può essere valutata all'inizio del ciclo di vita di sviluppo senza dover attendere la disponibilità di un sistema completo e funzionante.

Il testing di manutenibilità dinamico si concentra sulle procedure definite per manutenere una particolare applicazione (ad esempio, per eseguire aggiornamenti software), selezionando scenari di manutenzione come test case per garantire che i livelli di servizio richiesti siano ottenibili con dette procedure. Questo tipo di testing è particolarmente rilevante quando l'infrastruttura sottostante è complessa e le procedure di supporto possono coinvolgere più reparti/organizzazioni. Questa forma di testing può aver luogo all'interno del testing di accettazione operativa.

4.6.2 Sotto-caratteristiche di manutenibilità

La manutenibilità di un sistema può essere misurata in termini di effort richiesto per diagnosticare i problemi identificati all'interno di un sistema (analizzabilità) e testare il sistema modificato (testabilità). I fattori che influenzano l'analizzabilità e la testabilità comprendono l'impiego di buone pratiche di programmazione (ad esempio, commenti, denominazione di variabili, indentazione) e la disponibilità di documentazione tecnica (ad esempio, specifiche di progettazione del sistema, specifiche di interfaccia).

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

- Modificabilità: il grado con cui una componente o un sistema può essere modificato in modo efficace ed efficiente senza introdurre difetti o comportare il degrado della qualità del prodotto esistente
- Modularità: il grado con cui un sistema, prodotto o componente è composto da componenti separate, in modo tale che la modifica a una componente abbia un impatto minimo sulle altre
- Riusabilità: il grado con cui un asset può essere utilizzato in più di un sistema o nello sviluppo di altri asset

4.7 Test di portabilità

4.7.1 Introduzione

I test di portabilità si riferiscono in generale al grado con cui una componente software o un sistema possono essere trasferiti nell'ambiente di destinazione, la prima volta o partendo da un ambiente esistente.

L'[ISO25010] definisce le seguenti sotto-caratteristiche di portabilità:

- Installabilità: la capacità di un prodotto software di essere installato in un determinato ambiente
- Adattabilità: il grado con cui una componente o un sistema possono essere adattati per ambienti hardware e software diversi o in evoluzione
- Sostituibilità: la capacità di un altro prodotto software di essere utilizzato al posto di un determinato prodotto software per lo stesso scopo nello stesso ambiente

Il testing di portabilità può iniziare da componenti singole (ad esempio, la sostituibilità di una particolare componente, tipo il passaggio da un sistema di gestione di database a un altro) e quindi espandere l'ambito man mano che diventa disponibile più 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 di progettazione architetturale. Le review dell'architettura e della progettazione possono essere particolarmente produttive per identificare potenziali requisiti e problemi di portabilità (ad esempio, la dipendenza da un particolare sistema operativo).

4.7.2 Testing di installabilità

Il testing di installabilità viene eseguito sul software e sulle procedure utilizzate per installare il software nell'ambiente di destinazione. Ciò può includere, ad esempio, il software sviluppato per installare un sistema operativo su un processore o un "wizard" di installazione per installare un prodotto su un PC client.

Obiettivi tipici del testing di installabilità sono:

- Confermare che il software possa essere installato con successo seguendo le istruzioni fornite dal manuale di installazione (inclusa l'esecuzione di eventuali script di installazione) o utilizzando una procedura guidata. Questo include la verifica delle opzioni di installazione per

diverse configurazioni hardware/software e per diversi tipi di installazione (ad esempio, iniziale o di aggiornamento).
- Verificare se i failure che si verificano durante l'installazione (ad esempio, il mancato caricamento di particolari DLL) vengono gestiti correttamente dal software di installazione senza lasciare il sistema in uno stato indefinito (ad esempio, software parzialmente installato o configurazioni di sistema errate)
- Verificare se è possibile completare un'installazione/disinstallazione parziale
- Verificare se una procedura guidata di installazione è in grado di identificare correttamente piattaforme hardware o configurazioni non valide del sistema operativo
- Misurare se il processo di installazione può essere completato entro un numero specificato di minuti o entro un numero specificato di passi
- Confermare che il software possa essere correttamente disinstallato completamente o portato alla versione precedente ("downgrade")

Dopo il testing di installazione si esegue normalmente un testing delle funzionalità, per rilevare eventuali difetti introdotti dall'installazione (es. configurazioni errate, funzioni non disponibili). In parallelo al testing di installabilità viene normalmente eseguito un testing di usabilità (ad esempio, per confermare 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 di destinazione previsti (hardware, software, middleware, sistema operativo, ecc.). Un sistema "adattivo" è quindi un sistema che è in grado di adattare il proprio comportamento ai cambiamenti del proprio ambiente o di parti del sistema stesso. La specifica dei test di adattabilità richiede che le combinazioni degli ambienti di destinazione previsti siano identificate, configurate e disponibili al team di test. Questi ambienti vengono poi testati utilizzando una selezione di test case funzionali che attivano 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 verificare tale procedura.

I test di adattabilità possono essere eseguiti insieme ai test di installabilità e sono tipicamente seguiti da test funzionali per rilevare eventuali difetti che potrebbero essere stati introdotti nell'adattamento del software a un ambiente diverso.

4.7.4 Testing di sostituibilità

Il testing di sostituibilità si concentra sulla capacità delle componenti software di un sistema di essere rimpiazzate con altre. Ciò può essere particolarmente rilevante per i sistemi che utilizzano software commerciale off-the-shelf (COTS) per determinate componenti di sistema.

I test di sostituibilità possono essere eseguiti in parallelo ai test di integrazione funzionale quando sono disponibili componenti alternative per l'integrazione nel sistema completo. La sostituibilità può essere valutata mediante review tecnica o ispezione a livello di architettura e di progettazione, ponendo l'accento su una definizione chiara delle interfacce del sistema con potenziali componenti sostituibili.

4.8 Testing di compatibilità

4.8.1 Introduzione

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

- Coesistenza: il grado con cui un elemento sotto test può funzionare in modo soddisfacente insieme ad altri prodotti indipendenti in un ambiente condiviso. Questo aspetto è descritto di seguito.
- Interoperabilità: il grado con cui un sistema scambia informazioni con altri sistemi o componenti. Questo aspetto è descritto nel syllabus ISTQB Advanced Level Test Analyst [ISTQB_ALTA_SYL].

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 esempio, sullo stesso hardware) senza influenzare il comportamento degli altri (ad esempio, conflitti di risorse). I test di coesistenza vanno eseguiti 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 in cui è l'unica applicazione installata (dove i problemi di incompatibilità non sono rilevabili) e quindi distribuita su un altro ambiente (ad esempio, produzione) che esegue anche altre applicazioni.

Gli obiettivi tipici del testing di coesistenza includono:

- Valutazione del possibile impatto negativo sulla funzionalità del sistema quando le applicazioni vengono caricate nello stesso ambiente (ad es. utilizzo di risorse in conflitto quando un server esegue più applicazioni)
- Valutazione dell'impatto sulle applicazioni derivante dalla distribuzione di correzioni e aggiornamenti del sistema operativo

I problemi di coesistenza devono essere analizzati durante la pianificazione dell'ambiente target di produzione, ma i test effettivi vengono normalmente eseguiti dopo che è stato completato con successo il testing di sistema.

5. Review - 165 minuti

Parole chiave

anti-pattern

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 di una review è importante per il Technical Test Analyst

5.2 L'utilizzo delle Checklist nelle review

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

TTA-5.2.2 (K4) Analizzare una porzione di codice o pseudo-codice e identificare potenziali 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 una review tecnica e fornire le loro opinioni. Tutti i partecipanti alla revisione dovrebbero avere una formazione formale sulle review per comprendere meglio i propri ruoli e dovrebbero impegnarsi a contribuire all'ottenimento dei benefici di una revisione tecnica ben condotta. Questo include il mantenimento di un rapporto di lavoro costruttivo con gli autori nella descrizione e discussione dei commenti di review. Per una descrizione completa delle review tecniche, incluse numerose checklist, vedere [Wiegers02]. I Technical Test Analyst normalmente partecipano a review tecniche e ispezioni in cui portano un punto di vista operativo (comportamentale) che potrebbe essere mancante negli sviluppatori. Inoltre, i Technical Test Analyst svolgono un ruolo importante nella definizione, applicazione e manutenzione delle checklist di review e delle informazioni sulla gravità dei difetti.

Indipendentemente dal tipo di review da eseguire, al Technical Test Analyst deve essere concesso un tempo adeguato alla preparazione. Ciò include il tempo per esaminare l'oggetto della review, il tempo per controllare la documentazione con riferimenti incrociati per verificarne la coerenza, e il tempo per determinare cosa potrebbe mancare 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, l'individuazione di ciò che manca e la verifica che il prodotto descritto sia coerente con altri prodotti già sviluppati o in fase di sviluppo. Ad esempio, durante la revisione di un level test plan per il testing di integrazione, il Technical Test Analyst deve prendere in considerazione anche gli elementi che vengono integrati. Sono pronti per l'integrazione? Ci sono dipendenze che devono essere documentate? Sono disponibili i dati per testare i punti di integrazione? Una review non è isolata dal prodotto di lavoro della review. Deve anche considerare l'interazione di quell'elemento con gli altri nel sistema.

5.2 L'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 spersonalizzare la review, ad esempio "questa è la stessa checklist che utilizziamo per ogni review e non la stiamo utilizzando solo per la review tuo prodotto". Le checklist possono essere generiche e utilizzate per tutte le review o focalizzate su specifiche caratteristiche o aree di qualità. Per esempio, una checklist generica potrebbe verificare il corretto utilizzo dei termini "deve" e "dovrebbe", verificare la corretta formattazione e simili elementi di conformità. Una checklist mirata potrebbe invece 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 locale di sviluppo
  - Personale
  - Strumenti
  - Priorità
- La storia di precedenti successi e difetti
- Problemi particolari (ad esempio, efficienza delle prestazioni, sicurezza)

Le checklist dovrebbero essere personalizzate in base all'organizzazione e talvolta al progetto. Le checklist presentate in questo capitolo servono solo da esempio.

Alcune organizzazioni ampliano la nozione abituale di checklist includendo gli "anti-pattern" che si riferiscono a errori comuni, tecniche scadenti e altre pratiche inefficaci. Il termine deriva dal concetto popolare di "design pattern", ovvero soluzioni riutilizzabili per problemi comuni che si sono dimostrate

efficaci in situazioni pratiche [Gamma94]. Un anti-pattern, quindi, è un errore commesso comunemente e spesso implementato in qualità di utile scorciatoia.

È importante ricordare che se un requisito non è testabile, ovvero non è definito in modo tale che il Technical Test Analyst possa determinare come testarlo, allora è un difetto. Ad esempio, un requisito che dice "Il software deve essere veloce" non può essere testato. Come può il Technical Test Analyst determinare se il software è veloce? Se, invece, il requisito dice "Il software deve rispondere in un massimo di tre secondi in condizioni di carico specifiche", la testabilità di questo requisito è sostanzialmente migliore se sono definite le "condizioni di carico specifiche" (ad esempio, numero di utenti simultanei e attività eseguite dagli utenti). Questo è anche un requisito generale perché può generare molti test case in un'applicazione non banale. Anche la tracciabilità tra questo requisito e i test case è fondamentale, perché se il requisito dovesse cambiare, tutti i test case potrebbero dover essere rivisti e aggiornati secondo necessità.

5.2.1 Review architetturali

L'architettura del software rappresenta l'organizzazione fondamentale di un sistema, incarnata dalle sue componenti, dalle relazioni tra loro e con il contesto, e dai principi che ne governano la progettazione e l'evoluzione. [ISO42010], [Bass03].

Le checklist[1] utilizzate per le review architetturali potrebbero, ad esempio, includere la verifica della corretta implementazione dei seguenti elementi citati da [Web-2]:

- “Connection pooling: riduzione dell’overhead del tempo di esecuzione associato alla creazione di connessioni al database, stabilendo un pool condiviso di connessioni
- Bilanciamento del carico (load balancing): ripartizione uniforme del carico tra un insieme di risorse
- Elaborazione distribuita (distributed processing)
- Caching: utilizzo di una copia locale dei dati per ridurre il tempo di accesso
- Lazy instantiation
- Concorrenza delle transazioni (transaction concurrency)
- Isolamento dei processi tra Online Transactional Processing (OLTP) e Online Analytical Processing (OLAP)
- Replica dei dati”

5.2.2 Review del codice

Le checklist per le review del codice sono necessariamente molto dettagliate e, come le checklist per le revisioni architetturali, sono più utili quando sono specifiche per lingua, progetto e azienda. L'inclusione di anti-pattern a livello di codice è utile, in particolare per gli sviluppatori di software meno esperti.

Le checklist[1] utilizzate per le review del codice possono includere i seguenti elementi:

1. Struttura
    - Il codice realizza completamente e correttamente le specifiche di progetto?
    - Il codice è conforme agli standard di codifica pertinenti?
    - Il codice è ben strutturato, coerente nello stile e formattato in modo coerente?
    - Vi sono procedure non richiamate, non necessarie o codice non raggiungibile?
    - Nel codice sono rimasti stub o routine di test?
    - È possibile sostituire una porzione di codice con chiamate a componenti riutilizzabili esterne o funzioni di libreria?
    - Vi sono blocchi di codice ripetuto che potrebbero essere condensati in un'unica procedura?

[1] La domanda d'esame fornirà una porzione della checklist con cui rispondere alla domanda

- L'uso dello storage è efficiente?
- Vengono utilizzati simboli anziché constanti con "numeri magici" o costanti stringa?
- Vi sono dei moduli eccessivamente complessi che dovrebbero essere ristrutturati o suddivisi in più moduli?

2. Documentazione
- Il codice è chiaramente e adeguatamente documentato con uno stile di commento facile da manutenere?
- Tutti i commenti sono coerenti con il codice?
- La documentazione è conforme agli standard applicabili?

3. Variabili
- Tutte le variabili sono definite correttamente con nomi significativi, coerenti e chiari?
- Sono presenti variabili ridondanti o inutilizzate?

4. Operazioni aritmetiche
- Il codice evita di fare confronti di uguaglianza tra numeri in virgola mobile (floating-point)?
- Il codice previene sistematicamente gli errori di arrotondamento?
- Il codice evita addizioni e sottrazioni su numeri con ordini di grandezza molto diversi?
- I divisori sono testati per valori a zero o valori spuri?

5. Cicli e rami
- Tutti i cicli, rami e costrutti logici sono completi, corretti e annidati correttamente?
- 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 predefinito?
- Le condizioni di terminazione del ciclo sono ovvie e invariabilmente realizzabili?
- Gli indici o i pedici sono inizializzati correttamente, appena prima del ciclo?
- Le istruzioni racchiuse all'interno di cicli possono essere collocate all'esterno?
- Il codice nel ciclo evita di manipolare la variabile indice o di usarla all'uscita dal ciclo?

6. Programmazione difensiva
- Gli indici e i puntatori 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 in uscita?
- 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 controllata 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 automazione - 180 minuti

Parole chiave
cattura/riesecuzione, testing data-driven, debugging, emulatore, disseminazione dei guasti, hyperlink, testing keyword-driven, efficienza delle prestazioni, simulatore, esecuzione dei test, test management

Obiettivi di apprendimento per gli strumenti di test e automazione

6.1 Definizione del progetto di Test Automation

TTA-6.1.1 (K2) Riassumere le attività che il Technical Test Analyst esegue durante l'impostazione di un progetto di test automation
TTA-6.1.2 (K2) Riassumere le differenze tra l'automazione data-driven e quella keyword-driven
TTA-6.1.3 (K2) Riassumere i problemi tecnici più comuni che impediscono ai progetti di test automation di raggiungere il ritorno d'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 e l'iniezione dei guasti
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 web testing
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) Descrivere lo scopo degli strumenti utilizzati per supportare il testing delle applicazioni mobili

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 disegnati e progettati con cura. L'implementazione di una strategia di test automation senza una solida architettura si traduce solitamente in un insieme di strumenti costosi da mantenere, insufficienti per il raggiungimento degli obiettivi, e incapaci di raggiungere il ritorno atteso sull'investimento.

Un progetto di test automation va 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 e di integrazione dei componenti, così come svolgere 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.

Le attività che il Technical Test Analyst può eseguire per quanto riguarda l'esecuzione automatizzata dei test sono molteplici:

- Definizione di chi sarà responsabile dell'esecuzione dei test (eventualmente coordinandosi con un Test Manager)
- Selezione dello strumento più adatto in base all'organizzazione, alla tempistica, alle competenze del team e ai requisiti di manutenzione (va notato che questo potrebbe implicare la necessità di decidere di creare uno strumento da utilizzare piuttosto che acquisirne uno)
- Definizione dei requisiti di interfaccia tra lo strumento di automazione e altri strumenti quali quelli per la gestione dei test, la gestione dei difetti e gli strumenti utilizzati nell'integrazione continua
- Sviluppo di eventuali adattatori necessari per creare un'interfaccia tra lo strumento di esecuzione dei test e il software sottoposto a test
- Selezione dell'approccio automatizzato, ad esempio keyword-driven o data-driven (vedere la Sezione 6.1.1)
- Collaborare con il Test Manager per stimare il costo dell'implementazione, inclusa la formazione. Nei progetti Agile questo aspetto viene tipicamente discusso e concordato negli incontri di pianificazione di progetto/sprint con l'intero team.
- Pianificare il progetto di automazione e allocare il tempo per la manutenzione
- Formazione di Test Analyst e Business Analyst per l'utilizzo e la fornitura dei dati per l'automazione
- Definizione di come e quando verranno eseguiti i test automatizzati
- Definizione di come i risultati dei test automatizzati verranno combinati con i risultati dei test manuali

Nei progetti con una forte enfasi sulla test automation, molte di queste attività possono essere affidate a un Test Automation Engineer (per i dettagli vedere il programma del sillabo Advanced Level Test Automation Engineer [ISTQB_ALTAE_SYL]). Alcune delle attività organizzative possono essere prese in carico da un Test Manager, in base alle esigenze e alle preferenze del progetto. Nei progetti Agile l'assegnazione di queste attività ai singoli 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 all'automazione

Questa sezione prende in considerazioni alcuni fattori che influiscono sull'approccio alla test automation:

- Automatizzazione tramite la GUI
- Applicazione di un approccio data-driven

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

Il syllabus Advanced Level Test Automation Engineer [ISTQB_ALTAE_SYL] include ulteriori dettagli sulla selezione di un approccio di automazione.

6.1.1.1 Automazione tramite la GUI

La test automation non si limita al testing tramite la GUI. Esistono strumenti per aiutare ad automatizzare i test a livello di API, tramite un'interfaccia con la command line (CLI) e altri punti di interfaccia nel software da testare. 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 tramite la GUI è la tendenza della GUI a cambiare 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 può produrre test case automatizzati (spesso chiamati script di test) che non vengono più eseguiti come desiderato se la GUI cambia. Questo perché lo 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 gli script già registrati per riflettere tali modifiche.

Gli strumenti di cattura/riesecuzione possono essere comodi punti di partenza per lo sviluppo di script di automazione. Il tester registra una sessione di test e lo script registrato viene quindi modificato per migliorare la manutenibilità (ad esempio, sostituendo le sezioni nello script registrato con funzioni riutilizzabili).

6.1.1.2 Applicazione di un approccio data-driven

A seconda del software da testare, i dati utilizzati per ciascun test possono essere diversi sebbene i passi siano praticamente identici (ad esempio, testare la gestione degli errori per un campo di input inserendo più valori non validi e controllando l'errore restituito per ciascuno). Sviluppare e mantenere uno script di test automatizzato per ciascuno di questi valori non è efficiente. Una soluzione tecnica comune per questo problema è di spostare i dati dagli script a un archivio esterno quale un foglio di calcolo o un database. Si scrivono quindi funzioni per accedere ai dati specifici ad ogni esecuzione dello script di test, cosa che consente a un singolo script di lavorare attraverso una serie di dati di test che forniscono i valori di input e i valori dei risultati attesi (ad esempio, un valore mostrato in un campo di testo o un messaggio di errore). Questo approccio è chiamato data-driven.

Quando si utilizza questo approccio, oltre agli script che elaborano i dati forniti, sono necessarie harness e infrastruttura di supporto per l'esecuzione dello script o della serie 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. Nei progetti Agile, può anche essere coinvolto nella definizione dei dati il rappresentante aziendale (es. Product Owner), in particolare per i test di accettazione. Questa suddivisione del lavoro consente ai responsabili dello sviluppo degli script (ad esempio, il Technical Test Analyst) di concentrarsi sull'implementazione di script di automazione intelligenti mentre il Test Analyst mantiene la proprietà effettiva del test. Nella maggior parte dei casi, il Test Analyst sarà responsabile dell'esecuzione degli script di test una volta che l'automazione è stata implementata e testata.

6.1.1.3 Applicazione di un approccio keyword-driven

Un altro approccio, denominato keyword-driven o action word-driven, fa un ulteriore passo avanti separando anche l'azione da eseguire sui dati forniti dallo script di test [Buwalda01]. Per realizzare

questa ulteriore separazione, viene creato un metalinguaggio di alto livello che è descrittivo piuttosto che direttamente eseguibile. Le istruzioni di questo linguaggio descrivono un processo di business completo o parziale del dominio che può richiedere del testing. Ad esempio, le keyword potrebbero essere "Login", "CreateUser", e "DeleteUser". Una keyword descrive un'azione di alto livello che verrà eseguita nel dominio dell'applicazione. Si possono definire anche azioni di livello inferiore che denotano l'interazione con l'interfaccia software stessa, quali: “ClickButton”, “SelectFromList”, o “TraverseTree”, le quali possono essere utilizzate per testare le capacità della GUI che non si adattano perfettamente alle keyword di un processo di business.

Una volta definite le keyword e i dati da utilizzare, la figura che ha in carico l’automazione (il test automator, come ad esempio il Technical Test Analyst o il Test Automation Engineer) traduce le keyword dei processi di business e le azioni di livello inferiore in codice di test automation. Le keyword e le azioni, insieme ai dati da utilizzare, possono essere archiviate in fogli di calcolo o immesse utilizzando strumenti specifici che supportano la keyword-driven test automation. Il framework di test automation implementa la keyword come insieme di una o più funzioni o script eseguibili. Gli strumenti leggono i test case scritti con keyword e richiamano le corrispondenti funzioni o script di test che li implementano. Gli eseguibili sono implementati in modo altamente modulare per consentire una facile mappatura di keyword specifiche. Per implementare questi script modulari sono necessarie capacità di programmazione.

La separazione della logica di business dalla programmazione effettiva degli script consente un uso più efficace delle risorse di test. Il Technical Test Analyst, nel ruolo di figura che ha in carico l’automazione, può applicare efficacemente le proprie capacità di programmazione senza dover diventare un esperto di dominio in molti processi 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 d'investimento dell’automazione.

6.1.1.4 Gestione dei Fallimenti del software

In qualsiasi progetto di test automation è importante prevenire e gestire i failure del software. Se si verifica un failure, la figura che ha in carico l’automazione dei test deve determinare cosa deve fare il software. Il failure deve essere registrato e poi i test continuano? I test devono terminare? Il failure può essere gestito con un'azione specifica (ad es. un clic su un pulsante in una finestra di dialogo) o magari aggiungendo un ritardo nel test? I failure software non gestiti possono danneggiare i risultati dei test successivi e causare un problema con il test che era in esecuzione quando si è verificato il failure.

6.1.1.5 Considerare lo stato del sistema

Un’altra cosa importante è prendere in considerazione lo stato del sistema all'inizio e alla fine dei test. Potrebbe essere necessario assicurarsi che il sistema venga riportato a uno stato predefinito dopo il completamento dell'esecuzione del test. Questo consentirà di eseguire ripetutamente una suite di test automatici senza alcun intervento manuale per ripristinare il sistema a uno stato noto. Per fare ciò, la test automation potrebbe dover, ad esempio, 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 chiusura corretta (ovvero, disconnettendosi al termine dei test).

6.1.2 Modellizzare i processi di business per l’automazione

Per implementare un approccio keyword-driven per l'automazione dei test, 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 saranno i Test Analyst del progetto o, nel caso di progetti Agile, il rappresentante di business (ad esempio, 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 l'annullamento, la visualizzazione dell'ordine da annullare e la richiesta di conferma dell'annullamento. Le sequenze di keyword (ad es. "Login", "Select_Order", "Cancel_Order") e i dati di test pertinenti vengono utilizzati dal Test Analyst per specificare i test case. Di seguito è riportata una semplice tabella di input basata su keyword che potrebbe essere utilizzata per testare la capacità del software di aggiungere, reimpostare ed eliminare account utente:

Parola chiave	User	Password	Risultato
Add_User	User1	Pass1	Messaggio “aggiunto utente”
Add_User	@Rec34	@Rec35	Messaggio “aggiunto utente”
Reset_Password	User1	Welcome	Messaggio di conferma reset password
Delete_User	User1		Messaggio di username/password non valido
Add_User	User3	Pass3	Messaggio “aggiunto utente”
Delete_User	User2		Messaggio “utente no trovato”

Lo script di automazione che utilizza questa tabella cercherà i valori di input che lo script dovrà usare. Ad esempio, quando si arriva alla riga con la parola chiave "Delete_User", è richiesto solo il nome utente. Per aggiungere un nuovo utente sono richiesti sia il nome utente sia la password. È inoltre possibile fare riferimento ai valori di input da un archivio dati, come mostrato con la seconda parola chiave "Add_User", in cui viene immesso un riferimento ai dati anziché i dati stessi, cosa che fornisce maggiore flessibilità di accesso ai dati che potrebbero cambiare durante l'esecuzione dei test. Questo consente di combinare tecniche data-driven con lo schema a keyword.

Vi sono alcuni problemi che vanno presi in considerazione:

- Più granulari sono le keyword, più specifici sono gli scenari che possono essere coperti, ma il linguaggio di alto livello può diventare più complesso da mantenere.
- 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 legate direttamente alla GUI, è possibile che i test richiedano più manutenzione al verificarsi di modifiche.
- L'utilizzo di keyword aggregate può semplificare lo sviluppo ma complicare la manutenzione. Ad esempio, potrebbero esserci sei diverse keyword che creano collettivamente un record. Sarebbe meglio creare un’unica keyword che chiama tutte e sei le keyword consecutivamente per semplificare tale azione?
- Indipendentemente dalla quantità di analisi che è stata dedicata al linguaggio con cui vengono espresse le keyword, ci saranno momenti in cui saranno necessarie keyword nuove e diverse. Per ogni keyword ci sono due domini separati (cioè, la logica di business dietro di essa e la funzionalità di automazione per eseguirla). È pertanto necessario creare un processo per gestire entrambi i domini.

La keyword-based test automation può ridurre in modo significativo i costi di manutenzione dell'automazione dei test, ma è più costosa, più difficile da sviluppare e richiede più tempo di progettazione al fine di ottenere il previsto ritorno d'investimento.

Il syllabus Advanced Level Test Automation Engineer [ISTQB_ALTAE_SYL] include ulteriori dettagli sul tema della modellizzazione dei processi di business per l’automazione.

6.2 Strumenti specifici di test

Questa sezione contiene informazioni generali sugli strumenti che potrebbero essere utilizzati da un Technical Test Analyst oltre a quanto riportato nel syllabus Foundation Level [ISTQB_FL_SYL].

Informazioni dettagliate sui vari strumenti sono riportate anche nei seguenti syllabi ISTQB:

- Mobile Application Testing [ISTQB_FLMAT_SYL]
- Performance Testing [ISTQB_FLPT_SYL]
- Model-Based Testing [ISTQB_FLMBT_SYL]
- Test Automation Engineer [ISTQB_ALTAE_SYL]

6.2.1 Strumenti per la disseminazione e iniezione dei guasti

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

Gli strumenti di iniezione dei guasti (fault injection) forniscono deliberatamente input errati al software per garantire che il software sia in grado di affrontarli correttamente. Gli input vengono inseriti per interrompere il normale flusso di esecuzione del codice e consentire l'estensione della copertura del test (ad esempio, per coprire più condizioni di test negative e testare i meccanismi di gestione degli errori).

Entrambi i tipi di strumento sono generalmente utilizzati dal Technical Test Analyst, ma possono anche essere utilizzati dagli sviluppatori durante il test di codice nuovo.

6.2.2 Strumenti per i performance test

Gli strumenti per il performance test forniscono principalmente le seguenti funzioni:

- Generazione di carico
- Misurazione, monitoraggio, visualizzazione e analisi della risposta del sistema a un dato carico
- Informazioni dettagliate sul comportamento delle risorse del sistema e delle componenti di rete

La generazione del carico viene eseguita implementando uno script per un profilo operativo predefinito (vedere la Sezione 4.5.3). Lo script può essere inizialmente acquisito per un singolo utente (utilizzando se possibile uno strumento di cattura/riesecuzione) e quindi implementato per il profilo operativo specificato, utilizzando lo strumento di performance test. Questa implementazione deve tenere conto delle possibili variazioni dei dati per ogni transazione (o insieme di transazioni).

Gli strumenti generano il carico simulando un gran numero di utenti multipli (utenti "virtuali") che seguono i loro profili operativi per eseguire attività, inclusa la generazione di volumi specifici di dati di input. Rispetto ai singoli script di automazione, molti script di performance test riproducono l'interazione dell'utente con il sistema a livello di protocollo di comunicazione e non simulando l'interazione dell'utente tramite la GUI. Questo di solito consente di ridurre il numero di "sessioni" separate necessarie durante il test. Alcuni strumenti di generazione del carico sono in grado di pilotare l'applicazione utilizzando la sua interfaccia utente per misurare in modo più preciso vicino il tempo 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 e i report tipici sono:

- Numero di utenti simulati durante il test
- Numero e tipologia di transazioni generate dagli utenti simulati e tasso di arrivo delle transazioni
- Tempi di risposta a particolari richieste transazionali effettuate dagli utenti
- Report e grafici di carico rispetto ai tempi di risposta
- Report sull'utilizzo delle risorse (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:

- L'hardware e la larghezza di banda di rete necessaria per generare il carico
- La compatibilità dello strumento con il protocollo di comunicazione utilizzato dal sistema sotto test
- La flessibilità dello strumento per consentire una facile implementazione di diversi profili operativi
- I servizi di monitoraggio, analisi e reporting richiesti

Gli strumenti di performance test vengono generalmente acquistati e non sviluppati internamente a causa dell'effort di sviluppo richiesto. Tuttavia, può essere appropriato sviluppare uno strumento specifico se vi sono restrizioni tecniche che impediscono l'utilizzo di un prodotto commerciale o se il profilo di carico e i servizi da fornire sono relativamente semplici. Ulteriori dettagli sugli strumenti di performance test sono forniti nel syllabus Foundation Level Performance Testing [ISTQB_FLPT_SYL].

6.2.3 Strumenti per il testing web-based

Per il testing di applicazioni web sono disponibili strumenti specializzati sia open source che commerciali. Il seguente elenco mostra alcuni degli strumenti più comuni e il loro scopo:

- Strumenti di test di hyperlink per eseguire la scansione e controllare che su un sito web non vi siano link ipertestuali interrotti o mancanti
- Strumenti di controllo di HTML e XML per verificare la conformità agli standard HTML e XML delle pagine create da un sito web
- Simulatori di carico per verificare come reagirà il server in presenza di un gran numero di connessioni utente
- Strumenti di automazione 'lightweight' che funzionano con diversi browser
- Strumenti per eseguire la scansione del server, controllando i file orfani (non collegati)
- Correttori ortografici specifici per HTML
- Strumenti di controllo dei CSS (Cascading Style Sheet)
- Strumenti per verificare la presenza di violazioni degli standard, ad esempio gli standard di accessibilità della Sezione 508 negli Stati Uniti o M/376 in Europa
- Strumenti che rilevano problemi di sicurezza

Strumenti open source per il web testing possono essere recuperati tramite:

- Il 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.
- Il gruppo di lavoro sulla tecnologia delle applicazioni ipertestuali Web (WHATWG) [Web-5]. Questa organizzazione definisce gli standard HTML e fornisce uno strumento che esegue la convalida HTML [Web-6].

Alcuni strumenti che includono un web spider engine sono in grado di fornire informazioni sulla dimensione delle pagine e sul tempo necessario per scaricarle, oppure sulla presenza o meno di una pagina (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.

6.2.4 Strumenti a supporto del testing model-based

Il testing Model-Based (MBT) è una tecnica in base alla quale un modello formale, quale ad esempio una macchina a stati finiti, viene utilizzato per descrivere il comportamento del tempo di esecuzione previsto di un sistema controllato dal software. Gli strumenti commerciali per il MBT (vedere [Utting07]) spesso forniscono un motore che consente a un utente di "eseguire" il modello stesso. I thread di

esecuzione più interessanti possono essere salvati e utilizzati come test case. Il MBT è supportato anche da altri modelli eseguibili quali le Reti di Petri e gli Statechart.

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

Ulteriori dettagli sugli strumenti per il testing model-based si possono trovare nel syllabus Foundation Level Model-Based Testing [ISTQB_FLMBT_SYL].

6.2.5 Strumenti per il testing di componente e per le build

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

Gli strumenti per il testing di componente sono spesso specifici del linguaggio utilizzato. Ad esempio, se si usa Java, si può utilizzare JUnit per automatizzare gli unit test. Molti altri linguaggi hanno i propri strumenti specializzati di test; questi vengono denominati collettivamente 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 debugging facilitano il testing manuale di componente a un livello molto basso, consentendo agli sviluppatori e ai Technical Test Analyst di modificare i valori delle variabili durante l'esecuzione e di scorrere il codice riga per riga. Gli strumenti di debugging vengono utilizzati anche per aiutare lo sviluppatore a isolare e identificare i problemi nel codice quando viene segnalato un failure dal team di test.

Gli strumenti di automazione della build spesso consentono di attivare automaticamente una nuova build ogni volta che ne viene modificata una componente. Al termine della compilazione, i test di componente vengono eseguiti da altri strumenti. Questo livello di automazione per il processo di build si trova generalmente in ambienti di integrazione continua.

Se impostato correttamente, questo insieme di strumenti può avere un effetto molto positivo sulla qualità delle build rilasciate negli ambienti di test. Se una modifica apportata da un programmatore introduce difetti di regressione nella build, di solito farà fallire alcuni dei test automatizzati, innescando un'indagine immediata sulla causa dai failure prima che la build venga rilasciata nell'ambiente di test.

6.2.6 Strumenti per il supporto del test di applicazioni mobile

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

6.2.6.1 Simulatori

Un simulatore per applicazioni mobile modella l'ambiente di runtime 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 talvolta utilizzati come sostituti dei dispositivi reali durante il testing. Tuttavia, l'applicazione testata su un simulatore è diversa dall'applicazione che verrà distribuita.

6.2.6.2 Emulatori

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

Gli emulatori vengono spesso utilizzati per ridurre il costo degli ambienti di test sostituendo i dispositivi reali. Tuttavia, un emulatore non può sostituire completamente un dispositivo perché l'emulatore potrebbe comportarsi in modo diverso dal dispositivo mobile che tenta di imitare. Inoltre, alcune funzionalità quali il (multi)touch, l'accelerometro e altre potrebbero non essere supportate, in parte a causa dalle limitazioni della piattaforma utilizzata per eseguire l'emulatore.

6.2.6.3 Aspetti comuni

I simulatori e gli emulatori sono utili nella fase iniziale dello sviluppo poiché si integrano tipicamente con gli ambienti di sviluppo e consentono rapida distribuzione, testing e monitoraggio delle applicazioni. L'utilizzo di un emulatore o simulatore ne richiede l'avvio, l'installazione della necessaria app e quindi di testare la app come questa fosse sul dispositivo reale. Gli ambienti di sviluppo del sistema operativo mobile sono in genere forniti con il proprio emulatore e simulatore in bundle. Sono disponibili anche emulatori e simulatori di terze parti.

Gli emulatori e i simulatori consentono solitamente l'impostazione di vari parametri di utilizzo. Queste impostazioni riguardano, ad esempio, l'emulazione di rete a velocità diverse, l'intensità del segnale, la perdita di pacchetti, la modifica dell'orientamento, la generazione di interruzioni e dati di posizionamento GPS. Alcune di queste impostazioni possono essere molto utili perché difficili o costose da replicare con dispositivi reali, ad esempio la posizione globale GPS o l'intensità del segnale.

Ulteriori dettagli possono essere trovati nel syllabus Foundation Level Mobile Application Testing [ISTQB_FLMAT_SYL].

7. Riferimenti

7.1 Standard

Nei capitoli sotto indicati sono menzionati i seguenti standard:

- [RTCA DO-178C/ED-12C]: 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

- [ISO25010] ISO/IEC 25010 (2014) Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) System and software quality models
  Capitoli 2 e 4

- [ISO29119] ISO/IEC/IEEE 29119-4 International Standard for Software and Systems Engineering - Software Testing Part 4: Test techniques. 2015
  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

7.2 Documenti ISTQB

- [ISTQB_AL_OVIEW] Advanced Level Overview, Version 2019
- [ISTQB_ALSEC_SYL] Advanced Level Security Testing Syllabus, Version 2016
- [ISTQB_ALTAE_SYL] Advanced Level Test Automation Engineer Syllabus, Version 2017
- [ISTQB_FL_SYL] Foundation Level Syllabus, Version 2018
- [ISTQB_FLPT_SYL] Foundation Level Performance Testing Syllabus, Version 2018
- [ISTQB_FLMBT_SYL] Foundation Level Model-Based Testing Syllabus, Version 2015
- [ISTQB_ALTA_SYL] Advanced Level Test Analyst Syllabus, Version 2019
- [ISTQB_ALTM_SYL] Advanced Level Test Manager Syllabus, Version 2012
- [ISTQB_FLMAT_SYL] Foundation Level Mobile Application Testing Syllabus, 2019
- [ISTQB_GLOSSARY] Glossary of Terms used in Software Testing, Version 3.2, 2019

7.3 Libri

[Bass03] Len Bass, Paul Clements, Rick Kazman “Software Architecture in Practice (2nd edition)”, Addison-Wesley 2003, ISBN 0-321-15495-9

[Bath14] Graham Bath, Judy McKay, “The Software Test Engineer’s Handbook (2nd edition)”, Rocky Nook, 2014, ISBN 978-1-933952-24-6

[Beizer90] Boris Beizer, "Software Testing Techniques Second Edition", International Thomson Computer Press, 1990, ISBN 1-8503-2880-3
[Beizer95] Boris Beizer, “Black-box Testing”, John Wiley & Sons, 1995, ISBN 0-471-12094-4
[Burns18] Brendan Burns, “Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services”, O’Reilly, 2018, ISBN 13: 978-1491983645
[Buwalda01]: Hans Buwalda, “Integrated Test Design and Automation”, Addison-Wesley Longman, 2001, ISBN 0-201-73725-6
[Copeland03]: Lee Copeland, “A Practitioner's Guide to Software Test Design”, Artech House, 2003, ISBN 1-58053-791-X
[Gamma94] Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1994, ISBN 0-201-63361-2
[Jorgensen07]: Paul C. Jorgensen, “Software Testing, a Craftsman’s Approach third edition”, CRC press, 2007, ISBN-13:978-0-8493-7475-3
[Kaner02]: Cem Kaner, James Bach, Bret Pettichord; “Lessons Learned in Software Testing”; Wiley, 2002, ISBN: 0-471-08112-4
[Koomen06]: Tim Koomen, Leo van der Aalst, Bart Broekman, Michael Vroon, "TMap Next for result-driven testing"; UTN Publishers, 2006, ISBN: 90-72194-79-9
[McCabe76] Thomas J. McCabe, "A Complexity Measure", IEEE Transactions on Software Engineering, Vol. SE-2, No. 4, December 1976. PP 308-320
[McCabe96] Arthur H. Watson and Thomas J. McCabe. "Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric" (PDF), 1996, NIST Special Publication 500-235.
[NIST96] Arthur H. Watson and Thomas J. McCabe, "Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric", NIST Special Publication 500-235, Prepared under NIST Contract 43NANB517266, September 1996.
[Splaine01]: Steven Splaine, Stefan P. Jaskiel, “The Web-Testing Handbook”, STQE Publishing, 2001, ISBN 0-970-43630-0
[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

Nel seguito sono elencati riferimenti a informazioni disponibili in Internet. I riferimenti sono stati controllati al momento della pubblicazione, l’ISTQB non si assume responsabilità nel caso gli stessi risultino successivamente indisponibili.

[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://whatwg.org/validator/

Capitolo 4: [Web-1] [Web-4]
Capitolo 5: [Web-2]
Capitolo 6: [Web-3] [Web-5] [Web-6]

8. Appendice A: Panoramica delle caratteristiche di qualità

La seguente tabella permette di confrontare le caratteristiche di qualità descritte nell'ISO 9126 (usate nella versione 2012 del syllabus Technical Test Analyst) con quelle del più recente [ISO25010] (usate nella versione 2019 del syllabus). Sono evidenziate solo le caratteristiche rilevanti per il Technical Test Analyst.

ISO/IEC 25010	ISO/IEC 9126-1	Note
Efficienza delle prestazioni	**Efficienza**	
Comportamento nel tempo	Comportamento nel tempo	
Utilizzo delle risorse	Utilizzo delle risorse	
Capacità		Nuova sotto-caratteristica
Compatibilità		Nuova caratteristica
Coesistenza	Coesistenza	Spostata da Portabilità
Interoperabilità		Spostata da Funzionalità (Test Analyst)
Affidabilità	**Affidabilità**	
Maturità	Maturità	
Disponibilità		Nuova sotto-caratteristica
Tolleranza ai guasti	Tolleranza ai guasti	
Recuperabilità	Recuperabilità	
Sicurezza	**Sicurezza**	In precedenza senza sotto-caratteristiche
Confidenzialità		Nuova sotto-caratteristica
Integrità		Nuova sotto-caratteristica
Non-repudiation		Nuova sotto-caratteristica
Responsabilità		Nuova sotto-caratteristica
Autenticità		Nuova sotto-caratteristica
Manutenibilità	**Manutenibilità**	
Modularità		Nuova sotto-caratteristica
Riusabilità		Nuova sotto-caratteristica
Analizzabilità	Analizzabilità	
Modificabilità	Stabilità	Combinazione di modificabilità e stabilità
	Modificabilità	
Testabilità	Testabilità	
Portabilità	**Portabilità**	
Adattabilità	Adattabilità	
Installabilità	Installabilità	
	Coesistenza	Spostata in Compatibilità
Sostituibilità	Sostituibilità	
	Conformità	Rimossa nella 25010

9. Indice