

Certified Tester Advanced Level Test Analyst (CTAL-TA) Syllabus

v4.0

International Software Testing Qualifications Board



Traduzione Italiana Ver. B (13/07/2026)

Avviso di Copyright

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

Copyright © 2025 Gli autori della versione 4.0: Armin Born, Filipe Carlos, Wim Decoutere, István Forgács, Matthias Hamburg (Product Owner), Attila Kovács, Sandy Liu, François Martin, Stuart Reid, Adam Roman, Jan Sabak, Murian Song, Tanja Tremmel, Marc-Florian Wendland, Tao Xian Feng

Copyright © 2021-2022. Gli autori dell'aggiornamento v3.1.0, errata 3.1.1 ed errata 3.1.2: Wim Decoutere, István Forgács, Matthias Hamburg, Adam Roman, Jan Sabak, Marc-Florian Wendland.

Copyright © 2019 Gli autori dell'aggiornamento 2019: Graham Bath, Judy McKay, Jan Sabak, Erik van Veenendaal.

Copyright © 2012. Gli autori: Judy McKay, Mike Smith, Erik van Veenendaal.

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

- È possibile copiare estratti di questo documento per uso non commerciale, a condizione che venga riconosciuta la fonte. Qualsiasi Training Provider accreditato può utilizzare questo syllabus come base per un corso di formazione, a condizione che gli autori e ISTQB® siano riconosciuti come fonte e proprietari del copyright del syllabus e a condizione che qualsiasi pubblicità di tale corso di formazione possa menzionare il syllabus solo dopo aver ricevuto l'accreditamento ufficiale dei materiali di formazione da un Member Board riconosciuto da ISTQB®.
- Qualsiasi individuo o gruppo di individui può utilizzare questo syllabus come base per articoli e libri, a condizione che gli autori e ISTQB® siano riconosciuti come fonte e proprietari del copyright del syllabus.
- È proibito qualsiasi altro utilizzo di questo syllabus senza aver prima ottenuto l'approvazione scritta di ISTQB®.
- Qualsiasi Member Board riconosciuto da ISTQB® può tradurre questo syllabus a condizione che riproduca il suddetto avviso di copyright nella versione tradotta del syllabus.

Cronologia delle Revisioni

Versione	Data	Osservazioni
v4.0	02/05/2025	Aggiornamento principale con revisione generale e aggiornamento dell'ambito di applicazione
v3.1.2	31/01/2022	Errata: corretti piccoli difetti di formattazione, grammatica e formulazione.
v3.1.1	15/05/2021	Errata: l'avviso di copyright è stato adattato agli standard in vigore di ISTQB®.
v3.1	03/03/2021	Aggiornamento minore con riscrittura della sezione 3.2.3 e vari miglioramenti nella formulazione.
v3.0 (2019)	19/10/2019	Aggiornamento principale con revisione generale e riduzione dell'ambito di applicazione.
v2.0 (2012)	19/10/2012	Prima versione come Syllabus CTAL-TA separato.

Cronologia delle versioni della traduzione italiana del syllabus v4.0

Versione	Data	Osservazioni
Ver. B	13/07/2026	Revisione della traduzione con miglioramento della traduzione di alcuni termini e semplificazione della lettura del testo italiano
Ver. A	14/10/2025	Prima versione pubblicata della traduzione italiana

Indice dei Contenuti

Avviso di Copyright	2
Cronologia delle Revisioni	3
Indice dei Contenuti	4
Ringraziamenti	6
0 Introduzione	8
0.1 Scopo di questo Syllabus	8
0.2 Il Test Analyst Advanced Level nel Testing del Software	8
0.3 Percorso di Carriera per i Tester	8
0.4 Business Outcome	9
0.5 Obiettivi di Apprendimento e Livello Cognitivo delle Conoscenze	9
0.6 L'Esame per il Certificato Advanced Level Test Analyst	10
0.7 Accreditamento	10
0.8 Gestione degli Standard	10
0.9 Livello di Dettaglio	10
0.10 Com'è Organizzato questo Syllabus	11
1 I Compiti del Test Analyst nel Processo di Test – 225 minuti	13
Introduzione ai Compiti del Test Analyst nel Processo di Test	14
1.1 Testing nel Ciclo di Vita dello Sviluppo Software	14
1.2 Coinvolgimento nelle Attività di Test	14
1.2.1 Analisi dei Test	15
1.2.2 Progettazione dei Test	15
1.2.3 Implementazione dei Test	16
1.2.4 Esecuzione dei Test	17
1.3 Compiti Relativi ai Prodotti di Lavoro	17
1.3.1 Test Case di Alto Livello e Test Case di Basso Livello	18
1.3.2 Criteri di Qualità per i Test Case	18
1.3.3 Requisiti dell'Ambiente di Test	19
1.3.4 Identificazione degli Oracoli del Test	20
1.3.5 Requisiti dei Dati di Test	20
1.3.6 Sviluppo di Test Script utilizzando il Testing Keyword-Driven	21
1.3.7 Strumenti Applicati nella Gestione del Testware	22
2 I Compiti del Test Analyst nel Testing basato sul Rischio – 90 minuti	24
Introduzione ai Compiti del Test Analyst nel Testing basato sul Rischio	25
2.1 Analisi del Rischio	25
2.2 Controllo del Rischio	26
3 Analisi dei Test e Progettazione dei Test – 615 minuti	28
Introduzione all'Analisi dei Test e alla Progettazione dei Test	30
3.1 Tecniche di Test basate sui Dati	30
3.1.1 Testing di Dominio	30
3.1.2 Testing Combinatorio	31
3.1.3 Testing Casuale	32

3.2	Tecniche di Test basate sul Comportamento	33
3.2.1	Testing CRUD	33
3.2.2	Testing delle Transizioni di Stato	34
3.2.3	Testing basato su Scenari	35
3.3	Tecniche di Test basate su Regole	36
3.3.1	Testing della Tabella delle Decisioni	36
3.3.2	Testing Metamorfico	37
3.4	Testing basato sull'Esperienza	38
3.4.1	Test Charter a supporto del Testing Session-based	38
3.4.2	Checklist a supporto delle Tecniche di Test basate sull'Esperienza	39
3.4.3	Crowd Testing	40
3.5	Applicazione delle Tecniche di Test più Appropriate	41
3.5.1	Selezione delle Tecniche di Test per Mitigare i Rischi di Prodotto	41
3.5.2	Vantaggi e Rischi dell'Automazione della Progettazione dei Test	43
4	Caratteristiche di Qualità del Testing – 60 minuti	44
	Introduzione alle Caratteristiche di Qualità del Testing	45
4.1	Testing Funzionale	45
4.2	Testing di Usabilità	46
4.3	Testing di Flessibilità	47
4.4	Testing di Compatibilità	48
5	Prevenzione dei Difetti del Software – 225 minuti	50
	Introduzione alla Prevenzione dei Difetti del Software	51
5.1	Pratiche di Prevenzione dei Difetti	51
5.2	Supporto al Contenimento di Fase	52
5.2.1	Utilizzo di Modelli per Rilevare Difetti	52
5.2.2	Applicazione delle Tecniche di Review	53
5.3	Mitigazione della Ricorrenza dei Difetti	54
5.3.1	Analizzare i Risultati del Test per Migliorare il Rilevamento dei Difetti	54
5.3.2	Supportare la Root Cause Analysis con la Classificazione dei Difetti	55
6	Riferimenti	57
7	Appendice A – Obiettivi di Apprendimento/Livello Cognitivo di Conoscenza	62
8	Appendice B – Matrice di Tracciabilità dei Business Outcome rispetto agli Obiettivi di Apprendimento	64
9	Appendice C – Release Note	69
10	Appendice D – Elenco delle abbreviazioni	72
11	Appendice E – Termini specifici del dominio	73
12	Appendice F – Modello di Qualità del Software	74
13	Appendice G – Marchi commerciali	76
14	Indice analitico	77

Ringraziamenti

L'Assemblea Generale di ISTQB® ha pubblicato formalmente questo documento il 2 maggio 2025.

Il syllabus è stato redatto da un team di International Software Testing Qualifications Board: Armin Born, Filipe Carlos, Wim Decoutere, István Forgács, Matthias Hamburg (Product Owner), Attila Kovács, Sandy Liu, François Martin, Stuart Reid, Adam Roman, Jan Sabak, Murian Song, Tanja Tremmel, Marc-Florian Wendland, Tao Xian Feng. Il team ringrazia Julia Sabatine per la revisione, Gary Mogyorodi per la review tecnica, Daniel Pol'an per il costante supporto tecnico, il team di revisione e i Member Board per i suggerimenti e i contributi.

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

Edizione attuale (v4.0): Gergely Ágnes, Laura Albert, Dani Almog, Somying Atiporntham, Andre Baumann, Lars Bjørstrup, Ralf Bongard, Earl Burba, Filipe Carlos, Renzo Cerquozzi, Kanitta Chantaramanon, Alessandro Collino, Nicola De Rosa, Aneta Derkova, Dingguofu, Ole Chr. Hansen, Zheng Dandan, Karol Frühauf, Dinu Gamage, Chen Geng, Sabine Gschwandtner, Ole Chr. Hansen, Zsolt Hargitai, Tharushi Hettiarachchi, Ágota Horváth, Arnika Hryszko, Caroline Julien, Beata Karpinska, Ramit Manohar Kaul, Mattijs Kemmink, László Kvintovics, Thomas Letzkus, Marek Majerník, Donald Marcotte, Dénes Medzihradzky, Imre Mészáros, Krisztián Miskó, Gary Mogyorodi, Ebbe Munk, Maysinee Nakmanee, Ingvar Nordström, Tal Pe'er, Kunlanan Peetijade, Sahani Pinidiya, Lukáš Piška, Nishan Portoyan, Meile Posthuma, Yasassri Ratnayake, Randall Rice, Adam Roman, Marc Rutz, Jan Sabak, Vimukthi Saranga, Sumuduni Sasikala, Gil Shekel, Radoslaw Smilgin, Péter Sótér, Helder Sousa, Richard Taylor, Benjamin Timmermans, Giancarlo Tomasig, Robert Treffny, Shun Tsunoda, Stephanie Ulrich, François Vaillancourt, Linda Vreeswijk, Carsten Weise, Marc-Florian Wendland, Paul Weymouth, Elżbieta Wiśniewska, John Young, Claude Zhang.

Edizione 2021-2022 (v3.1): Gery Ágnes, Armin Born, Chenyifan, Klaudia Dussa-Zieger, Chen Geng (Kevin), Istvan Gercsák, Richard Green, Ole Chr. Hansen, Zsolt Hargitai, Andreas Hetz, Tobias Horn, Joan Killeen, Attila Kovacs, Rik Marselis, Marton Matyas, Blair Mo, Gary Mogyorodi, Ingvar Nordström, Tal Pe'er, Palma Polyak, Nishan Portoyan, Meile Posthuma, Stuart Reid, Murian Song, Péter Sótér, Lucjan Stapp, Benjamin Timmermans, Chris van Bael, Stephanie van Dijk, Paul Weymouth.

Successivamente, Tal Pe'er, Stuart Reid, Marc-Florian Wendland e Matthias Hamburg hanno suggerito miglioramenti formali, grammaticali e di formulazione che sono stati implementati e pubblicati nelle Errata 3.1.1 e 3.1.2.

Edizione 2019 (v3.0): Laura Albert, Markus Beck, Henriett Braunné Bokor, Francisca Cano Ortiz, Guo Chaonian, Wim Decoutere, Milena Donato, Klaudia Dussa-Zieger, Melinda Eckrich-Brajer, Péter Földházi Jr, David Frei, Chen Geng, Matthias Hamburg, Zsolt Hargitai, Zhai Hongbao, Tobias Horn, Ágota Horváth, Beata Karpinska, Attila Kovács, József Kreisz, Dietrich Leimsner, Ren Liang, Claire Lohr, Ramit Manohar Kaul, Rik Marselis, Marton Matyas, Don Mills, Blair Mo, Gary Mogyorodi, Ingvar Nordström, Tal Peer, Pálma Polyák, Meile Posthuma, Lloyd Roden, Adam Roman, Abhishek Sharma, Péter Sótér, Lucjan Stapp, Andrea Szabó, Jan te Kock, Benjamin Timmermans, Chris Van Bael, Erik van Veenendaal, Jan Versmissen, Carsten Weise, Robert Werkhoven, Paul Weymouth.

Edizione 2012 (v2.0): Graham Bath, Arne Becher, Rex Black, Piet de Roo, Frans Dijkman, Mats Grindal, Kobi Halperin, Bernard Homès, Maria Jönsson, Junfei Ma, Eli Margolin, Rik Marselis, Don Mills, Gary Mogyorodi, Stefan Mohacsi, Reto Mueller, Thomas Mueller, Ingvar Nordstrom, Tal Pe'er, Raluca Madalina Popescu, Stuart Reid, Jan Sabak, Hans Schaefer, Marco Sogliani, Yaron Tsubery, Hans Weiberg, Paul Weymouth, Chris van Bael, Jurian van der Laar, Stephanie van Dijk, Erik van Veenendaal, Wenqiang Zheng, Debi Zylbermann.

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

Revisione della traduzione italiana a cura di: Raffaele Bozzini, Alessandro Collino, Salvatore Reale, Cristina Sobrero

0 Introduzione

0.1 Scopo di questo Syllabus

Questo Syllabus costituisce la base per l'International Software Testing Qualification di Test Analyst Advanced Level. ISTQB® fornisce questo Syllabus come segue:

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

0.2 Il Test Analyst Advanced Level nel Testing del Software

La certificazione ISTQB® Advanced Level Test Analyst (CTAL-TA) fornisce le competenze necessarie per eseguire test software strutturati e approfonditi durante l'intero ciclo di vita dello sviluppo software. Descrive in dettaglio il ruolo e le responsabilità del test analyst in ogni fase di un processo di test standard e approfondisce importanti tecniche di test. La certificazione Advanced Level Test Analyst è rivolta a persone in possesso di un certificato Foundation Level che desiderano sviluppare ulteriormente la propria esperienza nell'analisi dei test e nelle tecniche di test.

In questo Syllabus, il test analyst è inteso come un ruolo che:

- si concentra sulle esigenze di business del cliente, più che sugli aspetti tecnici del testing
- esegue principalmente testing funzionale, ma contribuisce anche al testing non-funzionale incentrato sull'utente, come il testing di usabilità, adattabilità, installabilità o interoperabilità
- utilizza tecniche di test black-box e testing basato sull'esperienza, più che tecniche di test white-box
- migliora l'efficacia del testing utilizzando tecniche di prevenzione dei difetti

0.3 Percorso di Carriera per i Tester

Lo schema ISTQB® supporta i professionisti del testing in tutte le fasi della loro carriera. Le persone che ottengono la certificazione ISTQB® Advanced Level Test Analyst potrebbero anche essere interessate agli altri Core Advanced Levels (Technical Test Analyst e Test Management) e, successivamente, all'Expert Level (Test Management o Improving the Test Process). Chiunque desideri sviluppare competenze nelle pratiche di testing in un ambiente Agile potrebbe prendere in considerazione le certificazioni Agile Technical Tester o Agile Test Leadership at Scale. Inoltre, i percorsi specialistici offrono prodotti di certificazione incentrati su tecnologie e approcci di test specifici, caratteristiche di qualità e livelli di test

specifici o testing in particolari settori industriali. Visita il sito www.istqb.org per le ultime informazioni sullo schema di certificazione dei tester certificati ISTQB®.

0.4 Business Outcome

Questa sezione elenca i Business Outcome (Risultati di Business) attesi da un candidato che ha ottenuto la certificazione Advanced Level Test Analyst.

Un tester certificato Advanced Level Test Analyst è in grado di...

Codice	Descrizione
TA-BO1	Supportare ed eseguire il testing appropriato in base al ciclo di vita dello sviluppo software adottato.
TA-BO2	Applicare i principi del testing basato sul rischio.
TA-BO3	Selezionare e applicare tecniche di test appropriate per supportare il raggiungimento degli obiettivi del test
TA-BO4	Fornire documentazione con livelli adeguati di dettaglio e qualità
TA-BO5	Determinare i tipi appropriati di testing funzionale da eseguire
TA-BO6	Contribuire al testing non-funzionale
TA-BO7	Contribuire alla prevenzione dei difetti
TA-BO8	Migliorare l'efficienza del processo di test con l'uso di strumenti
TA-BO9	Specificare i requisiti per gli ambienti di test e i dati di test

0.5 Obiettivi di Apprendimento e Livello Cognitivo delle Conoscenze

Gli Obiettivi di Apprendimento (LO, Learning Objective) supportano i Business Outcome e vengono utilizzati per creare gli esami Certified Tester Advanced Level Test Analyst.

In generale, tutti i contenuti di questo Syllabus sono oggetto d'esame, ad eccezione dell'Introduzione e delle Appendici. Le domande d'esame verificheranno la conoscenza delle parole chiave a livello K1 (vedi sotto) o degli obiettivi di apprendimento al rispettivo livello di conoscenza.

Gli obiettivi di apprendimento specifici e i relativi livelli di conoscenza sono indicati all'inizio di ogni capitolo e classificati come segue:

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

Per tutti i termini elencati come le parole chiave appena sotto i titoli dei capitoli, è necessario ricordare il nome corretto e la definizione dal glossario ISTQB® (K1), anche se non esplicitamente menzionati in alcun obiettivo di apprendimento.

Ulteriori dettagli ed esempi di obiettivi di apprendimento sono riportati nella *Sezione 7*.

0.6 L'Esame per il Certificato Advanced Level Test Analyst

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

Fare riferimento al documento *'Exam Structures and Rules v1.1 Compatible with Syllabus Foundation and Advanced Levels and Specialists Modules'* per ulteriori dettagli relativi all'esame Test Analyst Advanced Level v4.0.

Il criterio di ammissione per sostenere l'esame ISTQB® Certified Tester Advanced Level Test Analyst è che i candidati abbiano un interesse per il testing del software. Tuttavia, è fortemente consigliato ai candidati anche di:

- Avere almeno un minimo di esperienza nello sviluppo o nel testing del software, come ad esempio sei mesi di esperienza come tester di sistema o di accettazione utente o come sviluppatore di software
- Frequentare un corso accreditato secondo gli standard ISTQB® (da uno dei Member Board riconosciuti dall'ISTQB).

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

0.7 Accredитamento

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

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

0.8 Gestione degli Standard

Organizzazioni internazionali di standardizzazione come IEEE e ISO hanno pubblicato standard associati alle caratteristiche di qualità e al testing del software. Tali standard sono citati in questo Syllabus. Lo scopo di questi riferimenti è quello di fornire un quadro di riferimento (come nei riferimenti alla norma ISO/IEC 25010 relativa alle caratteristiche di qualità) o di fornire una fonte di informazioni aggiuntive, se il lettore lo desidera. Si prega di notare che i syllabi ISTQB® utilizzano documenti standard come riferimenti. Il contenuto degli standard non è oggetto di esame. Per ulteriori informazioni sugli standard, consultare la sezione 6 - Riferimenti.

0.9 Livello di Dettaglio

Il livello di dettaglio di questo Syllabus consente corsi ed esami coerenti a livello internazionale. Per raggiungere questo obiettivo, il Syllabus è composto da:

- Obiettivi formativi generali che descrivono l'intento dell'Advanced Level Test Analyst
- Un elenco di termini che i candidati devono essere in grado di ricordare
- Obiettivi di apprendimento per ciascuna area di conoscenza, che descrivono i risultati di apprendimento cognitivi da conseguire
- Una descrizione dei concetti chiave, con riferimenti a fonti quali letteratura riconosciuta o standard

I contenuti del Syllabus non descrivono l'intera area di conoscenza del testing del software; riflettono il livello di dettaglio da trattare nei corsi di formazione per Advanced Level Test Analyst.

Il Syllabus utilizza la terminologia (cioè denominazione e significato) impiegata nel software testing e nella Quality Assurance, in conformità all'ISTQB® Glossary (ISTQB-Glossary, 2024).

Per la terminologia delle discipline correlate si rimanda ai rispettivi glossari: IREB-CPRE per l'ingegneria dei requisiti (IREB-Glossary, 2024) e IEEE-Pascal per l'ingegneria del software (Computer Society et al., 2024).

0.10 Com'è Organizzato questo Syllabus

Sono presenti cinque capitoli con contenuti d'esame. L'intestazione di primo livello di ciascun capitolo specifica il tempo assegnato al capitolo; non sono indicate tempistiche al di sotto del livello di capitolo. Per i corsi di formazione accreditati, il Syllabus richiede un minimo di **20,25 ore** di istruzione (**1215 minuti**), distribuite sui cinque capitoli come segue:

- Capitolo 1 (225 minuti): I compiti del Test Analyst nel processo di test
 - Lo studente apprende come il Test Analyst è coinvolto nei diversi cicli di vita dello sviluppo software.
 - Lo studente apprende come il Test Analyst è coinvolto nelle diverse attività di test.
 - Lo studente apprende i compiti svolti dal Test Analyst in relazione ai prodotti di lavoro.
- Capitolo 2 (90 minuti) I compiti del Test Analyst nel testing basato sul rischio
 - Lo studente apprende come il Test Analyst contribuisce all'analisi dei rischi di prodotto.
 - Lo studente apprende come analizzare l'impatto delle modifiche per determinare l'ambito del regression testing.
- Capitolo 3 (615 minuti) Analisi dei test e progettazione dei test
 - Lo studente apprende le tecniche di test basate sui dati, come il testing di dominio, il testing combinatorio e il testing casuale.
 - Lo studente apprende le tecniche di test basate sul comportamento, come il testing CRUD, il testing delle transizioni di stato e il testing basato su scenari.
 - Lo studente apprende le tecniche di test basate su regole, come il testing della tabella delle decisioni e il testing metamorfico.
 - Lo studente apprende il testing basato sull'esperienza, come il testing session-based e il crowd testing.

- Lo studente apprende come selezionare le tecniche di test appropriate per mitigare i rischi di prodotto.
- Capitolo 4 (60 minuti) Caratteristiche di qualità del testing
 - Lo studente apprende come eseguire diversi tipi di testing funzionale.
 - Lo studente apprende come utilizzare la conoscenza specifica della funzionalità per contribuire ai tipi di test non funzionali, come l'usability testing, il testing della flessibilità e il testing di compatibilità.
- Capitolo 5 (225 minuti) Prevenzione dei difetti del software
 - Lo studente apprende diverse pratiche di prevenzione dei difetti.
 - Lo studente apprende vari approcci che supportano il contenimento di fase.
 - Lo studente apprende come mitigare il ripresentarsi dei difetti.

1 I Compiti del Test Analyst nel Processo di Test – 225 minuti

Parole Chiave

ambiente di test, analisi dei test, ciclo di vita dello sviluppo software, condizione di test, dati di test, esecuzione dei test, keyword, implementazione dei test, oracolo del test, progettazione dei test, test analyst, test case, test case di alto livello, test case di basso livello, test script, testware

Obiettivi di Apprendimento per il Capitolo 1:

1.1 Testing nel Ciclo di Vita dello Sviluppo Software

TA-1.1.1 (K2) Riassumere il coinvolgimento del Test Analyst nei vari cicli di vita dello sviluppo software

1.2 Coinvolgimento nelle Attività di Test

TA-1.2.1 (K2) Riassumere i compiti svolti dal Test Analyst nell'ambito dell'analisi dei test

TA-1.2.2 (K2) Riassumere i compiti svolti dal Test Analyst nell'ambito della progettazione dei test

TA-1.2.3 (K2) Riassumere i compiti svolti dal Test Analyst nell'ambito dell'implementazione dei test

TA-1.2.4 (K2) Riassumere i compiti svolti dal Test Analyst nell'ambito dell'esecuzione dei test

1.3 Compiti Relativi ai Prodotti di Lavoro

TA-1.3.1 (K2) Distinguere tra test case di alto livello e test case di basso livello

TA-1.3.2 (K2) Spiegare i criteri di qualità per i test case

TA-1.3.3 (K2) Fornire esempi di requisiti dell'ambiente di test

TA-1.3.4 (K2) Spiegare il problema dell'oracolo del test e le possibili soluzioni

TA-1.3.5 (K2) Fornire esempi di requisiti per i dati di test

TA-1.3.6 (K3) Utilizzare il testing keyword-driven per sviluppare test script

TA-1.3.7 (K2) Riassumere le tipologie di strumenti per la gestione del testware

Introduzione ai Compiti del Test Analyst nel Processo di Test

Il Syllabus Certified Tester - Foundation Level (ISTQB CTFL) descrive due ruoli principali nel testing: un ruolo di test management e un ruolo di testing. In questo Syllabus, la persona che, nell'ambito del testing, è responsabile di eseguire il testing degli aspetti di business del software è chiamata Test Analyst (TA). Sebbene questa responsabilità sia raramente assegnata a una posizione o a un ruolo dedicato, il TA ha compiti chiaramente definiti e competenze richieste. In termini di livelli di test, il TA si concentra su testing di sistema, testing di accettazione e testing di integrazione dei sistemi. In termini di caratteristiche di qualità, le competenze del TA si focalizzano sull'adeguatezza funzionale e coprono anche alcune caratteristiche di qualità non funzionali orientate all'utente, come usabilità, adattabilità, installabilità e interoperabilità.

1.1 Testing nel Ciclo di Vita dello Sviluppo Software

L'organizzazione delle attività di test può variare a seconda del ciclo di vita dello sviluppo software (SDLC) adottato. Di conseguenza, il coinvolgimento del TA nelle attività di test può variare in base al modello di SDLC scelto.

Nei **modelli di sviluppo sequenziali**, le attività di sviluppo sono suddivise in fasi e iniziano quando la fase precedente è completata. In genere, esiste poca sovrapposizione tra queste attività. Pertanto, i compiti del TA di solito cambiano nel tempo. Nelle fasi iniziali dell'SDLC, il TA si focalizza sul supporto alla pianificazione dei test. Il TA inizia l'analisi dei test quando viene prodotta la base di test. La progettazione dei test e l'implementazione dei test vengono svolte in parallelo alla progettazione e all'implementazione del software. Infine, il TA esegue i test e supporta il completamento dei test nelle fasi finali dell'SDLC.

Nei **modelli di sviluppo incrementali**, il software è suddiviso in incrementi più piccoli e gestibili. Ogni incremento è sviluppato e testato in modo indipendente. Di conseguenza, il TA svolge le stesse attività per ciascun incremento (cioè analisi dei test, progettazione dei test, implementazione dei test, esecuzione dei test e supporto al test management). Tuttavia, il lavoro del TA può essere organizzato diversamente per ogni incremento. Il testing si focalizza sulle funzionalità nuove o modificate. Inoltre, a causa del rischio crescente di regressione, il TA deve prestare particolare attenzione al refactoring e alla composizione delle suite di regression testing.

Nei **modelli di sviluppo iterativi**, il processo di sviluppo è ciclico. Il progetto attraversa cicli ripetuti di prototipazione, testing, affinamento e rilascio. Il ruolo del TA è dinamico e adattivo. Il TA collabora strettamente con gli sviluppatori e i rappresentanti del business, adattandosi al prodotto in evoluzione. Il TA adegua e modifica le condizioni di test e i test case man mano che il software evolve, e fornisce feedback per migliorare il processo di test a ogni iterazione. Quanto più frequenti sono le iterazioni, tanto più critiche diventano l'implementazione e la manutenzione continuativa dei regression test da parte del TA.

Un SDLC può combinare elementi di vari modelli, tecniche e approcci specifici (ad esempio, lo sviluppo software Agile combina aspetti sia dei modelli iterativi sia di quelli incrementali). In tali casi, il coinvolgimento del TA dipenderà dalle caratteristiche specifiche dell'SDLC e da come sono combinate. Una buona pratica comune a tutti i modelli di SDLC è che il TA dovrebbe essere coinvolto fin dalle fasi iniziali dell'SDLC.

1.2 Coinvolgimento nelle Attività di Test

Il Syllabus Certified Tester Foundation Level (ISTQB CTFL) descrive sette attività all'interno del processo

di test.

Il TA è principalmente focalizzato su quattro di esse: analisi dei test, progettazione dei test, implementazione dei test ed esecuzione dei test. I compiti del TA in queste quattro attività sono descritti in dettaglio nelle sezioni seguenti.

1.2.1 Analisi dei Test

Durante l'analisi dei test, il TA verifica la completezza della base di test e raccoglie ogni informazione aggiuntiva rilevante per il testing. Questo include non solo documentazione ma anche informazioni verbali, p.e. conversazioni nella scrittura collaborativa delle user story (vedi ISTQB CTFL, par. 4.5.1). Le modifiche alla base di test possono portare ad adeguamenti dell'ambito del test in coordinamento con il test management.

Per procedere in modo efficace con l'analisi dei test, il TA verifica i seguenti criteri di ingresso:

- È stata eseguita la pianificazione dei test e sono chiari l'ambito del test, gli obiettivi del test e l'approccio del test.
- La base di test (che contiene informazioni come requisiti o user story) è definita.
- I rischi di prodotto già identificati sono stati valutati e, se richiesto, documentati.

Il TA valuta la base di test per identificare eventuali difetti e per valutarne la testabilità, fornendo così un riscontro anticipato ai product owner. Questo può includere la modellazione del comportamento del sistema in base alle tecniche di test da applicare (vedi [Sezione 3](#), [Sezione 5.2.1](#)). Anche le tecniche di review vengono applicate come parte del processo (vedi [Sezione 5.2.2](#)). Se non corretti direttamente, i difetti relativi alla base di test devono essere documentati. Inoltre, il TA determina gli oracoli del test necessari (vedi [Sezione 1.3.4](#)).

Il TA definisce e assegna una priorità le condizioni di test per ogni elemento di test in ambito. Le condizioni di test indirizzano gli obiettivi del test (vedi ISTQB CTFL, par. 1.1.1) e devono essere tracciabili rispetto agli elementi della base di test. L'ambito e il focus delle condizioni di test tengono conto dei rischi di prodotto. Nei modelli di sviluppo incrementali o iterativi, questo include la definizione dell'ambito del regression testing sulla base di un'analisi degli impatti. Nello sviluppo software Agile, le condizioni di test possono essere espresse come criteri di accettazione che riflettono i rischi delle user story.

Il TA può procedere per passi, partendo dalle condizioni di test di alto livello come "funzionalità della schermata x". In seguito, il TA definisce condizioni di test più dettagliate (di basso livello) come "la schermata x rifiuta numeri di conto corrente con una cifra in meno". Questo approccio supporta una copertura sufficiente e consente un inizio anticipato della progettazione dei test, p.e. per user story che necessitano ancora di essere raffinate.

Il TA coinvolge gli stakeholder nella review delle condizioni di test, per assicurare che la base di test sia chiaramente compresa e che il testing sia allineato agli obiettivi del test.

1.2.2 Progettazione dei Test

La progettazione dei test descrive come eseguire il testing per raggiungere gli obiettivi del test dichiarati. Questo viene tipicamente ottenuto tramite i test case. Il modo in cui si esegue la progettazione dei test dipende da molti fattori, inclusi la copertura richiesta, la base di test, l'SDLC, i vincoli di progetto e la conoscenza/esperienza dei tester.

Durante la progettazione dei test, il TA determina in quali aree è più appropriata la progettazione di test case di basso livello o di test case di alto livello (vedi [Sezione 1.3.1](#)). In entrambi i casi, il TA deve

identificare chiari criteri di superamento/fallimento. Il TA progetta i test case per le condizioni di test nuove o modificate in base ai criteri di qualità (vedi [Sezione 1.3.2](#)). Per i regression test, generalmente è sufficiente la selezione di test case di alto livello esistenti o l'adattamento di test case di basso livello esistenti sulla base della loro priorità.

Il TA aggiorna la tracciabilità tra la base di test, le condizioni di test e i test case. Nel testing basato sull'esperienza, i test case non sono sempre documentati; in alternativa, le condizioni di test (tra le altre cose) possono guidare l'esecuzione dei test. Il TA può progettare alcuni test case a partire dagli obiettivi dei test di alto livello.

Oltre a questi compiti, il TA definisce i requisiti dell'ambiente di test (vedi [Sezione 1.3.3](#)), identifica, crea e specifica i requisiti per i dati di test (vedi [Sezione 1.3.5](#)).

Il TA utilizza i criteri di uscita definiti nella pianificazione dei test per determinare quando è stato progettato un numero sufficiente di test case. Tuttavia, altri criteri di uscita, come i livelli di rischio residuo o i vincoli di progetto (p.e. budget o tempi), possono indicare quando concludere la progettazione dei test.

La progettazione dei test può essere supportata da strumenti, ma dovrebbe rimanere indipendente rispetto a strumenti e tecnologie. La progettazione dei test applica un approccio sistematico utilizzando tecniche di test o, in alternativa, viene svolta ad hoc.

I test case hanno un ruolo comunicativo e dovrebbero risultare comprensibili agli stakeholder rilevanti. Poiché un test case potrebbe non essere sempre eseguito dal suo autore, altri tester devono essere in grado di comprendere come eseguirlo, quali sono gli obiettivi del test (cioè le condizioni di test sottostanti) e la sua importanza. I test case devono inoltre essere comprensibili agli sviluppatori, che potrebbero implementare i test o rieseguirli in caso di failure, e agli auditor, che potrebbero doverne valutare l'approvazione.

1.2.3 Implementazione dei Test

Durante l'implementazione dei test, il TA fornisce il testware necessario per l'esecuzione dei test. Il TA può organizzare le test procedure e i test script in test suite, o proporre test case per la test automation. La definizione delle test procedure richiede l'identificazione accurata di vincoli e dipendenze che possono influenzare l'ordine di esecuzione. Oltre ai passi contenuti nei test case, le test procedure includono passi per impostare eventuali precondizioni iniziali (p.e. caricamento dei dati di test da un repository di dati), verificare risultati attesi e post-condizioni, ed eseguire i passi per il ripristino dopo l'esecuzione (p.e. reset di database, ambiente e sistema).

Il TA prioritizza le test procedure e i test script per l'esecuzione dei test, in base ai criteri di prioritizzazione identificati durante l'analisi del rischio e la pianificazione dei test, e identifica le test procedure o i test script da eseguire sulla versione corrente dell'oggetto di test. Questo consente di eseguire insieme, in un test run specifico, i test correlati (p.e. per nuove funzionalità o per il regression testing). Il TA aggiorna la tracciabilità tra la base di test e gli altri elementi del testware come le test procedure, i test script e le test suite.

Il TA può assistere il TM nella definizione di una schedulazione dell'esecuzione dei test, inclusa l'allocazione delle risorse, per consentire un'esecuzione efficiente dei test definendone l'ordine di esecuzione (vedi ISTQB CTFL par. 5.1.5).

Il TA crea i dati di input e dell'ambiente da caricare nei database e in altri repository (vedi [Sezione 1.3.5](#)). Questi dati devono essere "fit for purpose" (adatti allo scopo) per supportare gli specifici obiettivi del test.

Il TA dovrebbe anche verificare che l'ambiente di test sia completamente predisposto e pronto per l'esecuzione dei test (vedi [Sezione 1.3.3](#)). La modalità migliore consiste nel progettare ed eseguire uno smoke testing. L'ambiente di test dovrebbe rilevare difetti nell'oggetto di test attraverso l'esecuzione dei

test, operare normalmente quando non si verificano failure e, se richiesto, replicare adeguatamente l'ambiente di produzione o dell'utente finale.

Il livello di dettaglio e la complessità delle attività svolte durante l'implementazione dei test possono essere influenzati dal livello di dettaglio delle condizioni di test e dei test case. In alcuni casi si applicano normative e standard, e il testware dovrebbe fornire evidenze di conformità agli standard applicabili (p.e. [RTCA DO-178C](#), 2011).

1.2.4 Esecuzione dei Test

L'esecuzione dei test viene svolta secondo la schedulazione definita. Le attività tipiche svolte dal TA sono l'esecuzione dei test, il confronto tra i risultati effettivi e i risultati attesi, l'analisi delle anomalie, la segnalazione dei difetti e il logging dei risultati dei test.

Il TA esegue i test manualmente. Questo include il testing esplorativo, l'esecuzione delle test procedure, il regression testing e il testing confermativo. Per il testing esplorativo, il TA può utilizzare il testing session-based con test charter (vedi [Sezione 3.4.1](#)). Il TA può anche eseguire test script automatizzati, ma l'esecuzione e l'analisi in caso di failure possono essere un compito di sviluppatori, test automation engineer (TAE) o technical test analyst (TTA).

Il TA analizza le anomalie che si verificano durante le esecuzioni manuali o automatizzate per stabilirne le cause probabili. Un'anomalia può essere la conseguenza di un difetto nell'oggetto di test. Tuttavia, possono esistere anche altre cause, tra cui precondizioni mancanti, dati di test errati, difetti nei test script o nell'ambiente di test, o un'errata comprensione delle specifiche. Il TA registra i risultati effettivi dell'esecuzione, comunica i difetti sulla base dei failure osservati e, se necessario, ne redige report.

Il TA aggiorna la tracciabilità tra la base di test e altro testware tenendo conto dei risultati dei test. Queste informazioni consentono di trasformare i risultati dei test in informazioni di alto livello sul rischio o sulla copertura, che permettono agli stakeholder di prendere decisioni informate. Ad esempio, chiariscono agli stakeholder quanti test case relativi a una condizione di test sono stati superati o falliti.

Oltre a queste attività tipiche, il TA valuta i risultati dei test, includendo i seguenti compiti:

- Riconoscere cluster di difetti, che possono indicare la necessità di ulteriore testing su una particolare parte dell'oggetto di test (vedi [Sezione 5.3.1](#)).
- Rieseguire manualmente i test automatizzati falliti, per assicurarsi che la test automation non abbia prodotto un risultato falso-positivo.
- Suggestire test aggiuntivi sulla base di quanto appreso dai test precedenti.
- Identificare nuovi rischi dalle informazioni ottenute durante l'esecuzione dei test.
- Suggestire miglioramenti alla progettazione dei test o all'implementazione dei test (p.e. miglioramenti alle test procedure) o anche al System Under Test (SUT).
- Suggestire miglioramenti alle suite di regression test, inclusi refactoring, adeguamenti dell'ambito e test automation (vedi [Sezione 2.2](#)).

1.3 Compiti Relativi ai Prodotti di Lavoro

Il TA deve assicurare la qualità dei prodotti di lavoro di cui è responsabile. Questo include test case, ambienti di test, dati di test, oracoli del test e test script. In questo paragrafo, il Syllabus tratta i compiti del TA relativi al testware e ai tipi di strumenti dedicati alla gestione del testware.

1.3.1 Test Case di Alto Livello e Test Case di Basso Livello

Un test case di alto livello (*high-level test case*, detto anche *test case astratto* o *test case logico*) descrive le circostanze in cui l'oggetto di test viene esaminato indicando quali condizioni di test sono coperte dal test case. I test case di alto livello sono quindi adatti a garantire che i test coprano tutte le condizioni di test rilevanti. I test case di alto livello non contengono informazioni concrete su precondizioni, dati di input, risultati attesi o post-condizioni: tutto è espresso a livello astratto (p.e. "ordinare più di un libro, con un prezzo totale dell'ordine che generi uno sconto; risultato atteso: viene applicato lo sconto").

Un test case di basso livello (*low-level test case*, detto anche *test case concreto* o *test case fisico*) è il perfezionamento dettagliato di un test case di alto livello. I test case di basso livello descrivono quali dati devono essere preparati, quali azioni deve eseguire il tester (se necessario) e quale sia il risultato atteso concreto. I test case di basso livello contengono precondizioni, dati di input, risultati attesi e post-condizioni specifici (p.e. "ordinare i libri B1 (10 \$) e B2 (20 \$), con un prezzo totale di 30 \$; risultato atteso: sconto applicato: 10%, prezzo totale: 27 \$").

Tipicamente, un TA progetta inizialmente test case di alto livello, che costituiscono la base per sviluppare test case di basso livello. Un test case di alto livello può essere implementato in uno o più test case di basso livello. Talvolta, il test case può rimanere di alto livello, con le informazioni concrete determinate dal TA durante l'esecuzione dei test. Ad esempio, i test case di alto livello possono guidare il TA nella creazione degli obiettivi del test all'interno di un test charter per il testing session-based, consentendo al TA di dettagliare tali obiettivi durante l'esecuzione dei test (vedi [Sezione 3.4.1](#)).

Il passaggio da test case di alto livello a quelli di basso livello non è solo un riempimento di valori concreti: è anche un passaggio dal concettuale al tecnico. Spesso è rimandato dalla progettazione dei test all'implementazione dei test, soprattutto se sono necessari specifici dati di test. Il TA deve assicurare che sia conosciuto tutto ciò che è necessario per eseguire i test case di basso livello (Koomen et al., 2006). Nella pratica, molti test case sono ibridi, concreti in alcuni aspetti e astratti in altri, spesso a causa di un compromesso tra manutenibilità e comprensibilità dei test case.

1.3.2 Criteri di Qualità per i Test Case

Trascurare la qualità dei test case può portare a molti problemi, come alti costi di manutenibilità, minore comprensibilità o ritardi di esecuzione. I criteri di qualità per i test case sono un primo passo verso test case più manutenibili. Includono:

- **Correttezza.** Un test case deve facilitare la verifica accurata delle condizioni di test su cui è basato.
- **Fattibilità.** Deve essere possibile eseguire un test case.
- **Necessità.** Ogni test case dovrebbe coprire un chiaro obiettivo del test, come espresso nel suo titolo o sommario. Evitare duplicati. Non dovrebbero essere progettati test case per quello che non deve essere testato.
- **Comprensibilità.** I test case possono essere sottoposti a review, modificati ed eseguiti da persone diverse dall'autore. Il TA dovrebbe scrivere i test case in un linguaggio e formato comprensibili a tutti gli stakeholder coinvolti, senza spiegare l'ovvio. I test case complessi dovrebbero essere semplificati o suddivisi.
- **Tracciabilità.** I test case dovrebbero essere tracciabili verso le condizioni di test, i requisiti e i rischi, per consentire al TA di mantenerli aggiornati (vedi ISTQB CTFL par. 1.4.4).
- **Consistenza.** Una consistenza nel linguaggio, nella formattazione e nella struttura rende i test case più facili da comprendere e mantenere. A tale scopo, il TA può usare un glossario tale scopo.
- **Precisione.** Dovrebbe esistere un'unica interpretazione di un test case per evitare risultarti falsi-

negativi e falsi-positivi. Termini ambigui come “adeguato”, “se necessario” o “diversi” andrebbero evitati.

- **Completezza.** Dovrebbero essere presenti tutti gli attributi necessari (vedi [ISO/IEC/IEEE 29119-3](#), 2021), inclusi i dati di test richiesti (vedi [Sezione 1.3.5](#)) e un risultato atteso chiaro, per evitare dubbi durante il confronto con il risultato effettivo.
- **Sinteticità.** La granularità dei test case (cioè un singolo test case ampio con molte azioni di test, rispetto a test case più piccoli) dovrebbe corrispondere alla base di test e alle condizioni di test. Sono preferibili test case più piccoli, focalizzati su pochi elementi di copertura, perchè facilitano la rilevazione delle cause dei failure, possono essere combinati in modo flessibile in test procedure e test suite, e un failure durante l'esecuzione non bloccherebbe ulteriori test.

Il formato e il livello di dettaglio dei test case dipendono dal contesto di progetto e di prodotto e dovrebbero essere concordati all'interno del team di test.

1.3.3 Requisiti dell'Ambiente di Test

L'ambiente di test è un fattore critico di successo per l'esecuzione dei test sia manuale sia automatizzata. L'implementazione dell'ambiente di test influisce sulla testabilità, sulla rilevazione dei difetti, sui costi complessivi del testing e sull'affidabilità dei risultati del test. Un test case che passa o fallisce nell'ambiente di test dovrebbe avere lo stesso esito se eseguito in produzione. Idealmente, un ambiente di test è robusto, prevedibile e integrato con il framework di test automation, ove richiesto. Il TA può definire i requisiti dell'ambiente di test durante la progettazione dei test, sulla base dell'analisi di:

- condizioni di test, test case e requisiti sui dati di test, in modo che i requisiti dell'ambiente descrivano le condizioni necessarie per predisporre e mantenere l'ambiente, in modo da garantire le precondizioni dell'esecuzione dei test;
- livelli di test e tipi di test, che influenzano il compromesso tra flessibilità dell'ambiente di test e similarità con l'ambiente di produzione;
- disponibilità e indipendenza di componenti e sistemi, che possono indicare la necessità di usare i test double (p.e. stub o driver).

I requisiti dell'ambiente di test descrivono gli elementi dell'ambiente di test, classificabili in più categorie, come hardware, middleware, software, servizi virtualizzati, rete, interfacce, strumenti, sicurezza, configurazione e luogo. Per ogni elemento, i requisiti dovrebbero includere ([ISO/IEC/IEEE 29119-3](#), 2021):

- identificativo univoco – usato per scopi di tracciabilità;
- descrizione – con dettaglio sufficiente a implementarlo come richiesto;
- responsabilità – chi è responsabile di renderlo disponibile;
- periodo necessario – quando e per quanto tempo è richiesto l'elemento;
- fedeltà – grado secondo cui questo elemento rappresenta o si discosta dall'ambiente di produzione.

I requisiti dovrebbero inoltre indirizzare le richieste generali dell'ambiente di test, inclusi setup, backup e restore, esigenze di sicurezza, possibilità di modificare l'ambiente, ruoli e autorizzazioni (Koomen et al., 2006).

Il TA dovrebbe documentare i requisiti in modo chiaro, compatto e consistente; può utilizzare diagrammi o tabelle. Per evitare ridondanze e documentazione superflua, il TA può riferirsi ad ambienti di test esistenti e focalizzarsi sulle necessità specifiche del livello di test. Anche gli stakeholder rilevanti (p.e. sviluppatore, TTA, TAE, business analyst, sponsor e product owner) dovrebbero eseguire la review, approvare e aggiornare i requisiti dell'ambiente di test.

1.3.4 Identificazione degli Oracoli del Test

Un oracolo del test è necessario per determinare i risultati attesi nel testing dinamico. Idealmente, la base di test fornirà gli oracoli (p.e. in una specifica testuale o formale). Anche l'esperienza o la conoscenza umana dell'oggetto di test può fungere da oracolo. Un oracolo del test automatizzato può essere richiesto per ragioni di costi-efficacia (p.e. se gli input sono generati automaticamente o gli oracoli umani sono costosi).

In base alla qualità e alla completezza della base di test o delle caratteristiche del sistema, un oracolo con un adeguato rapporto tra costi ed efficacia potrebbe non essere disponibile. Questo è noto come *problema dell'oracolo del test*. Fattori tipici che contribuiscono al problema dell'oracolo del test includono la complessità legata ai dati, il non-determinismo (p.e. sistemi AI-based), il comportamento probabilistico e requisiti mancanti o ambigui.

Alcune soluzioni al problema dell'oracolo del test sono le seguenti (Barr et al., 2014):

- Pseudo-oracolo: sistemi sviluppati a parte che soddisfano la stessa specifica dell'oggetto di test (p.e. sistemi legacy, versioni semplificate dell'oggetto di test). Sviluppare un pseudo-oracolo dedicato per un oggetto complesso può essere costoso, ma è tipico per sistemi critici.
- Testing model-based: può formalizzare l'oracolo come parte del modello di test. Consente la generazione dei risultati attesi e la derivazione dei test dal modello. Le tecniche di test basate sul comportamento implicano spesso l'implementazione di un oracolo automatizzato (vedi [Sezione 3.2.2](#) e [Sezione 3.2.3](#)).
- Testing property-based: utilizza proprietà specificate dell'oggetto di test per verificare le relazioni tra input e risultato atteso dei singoli test case. Se tale relazione non è soddisfatta, il test case fallisce. Questa soluzione è adatta alla test automation, ma la sua efficacia dipende dalle relazioni, che possono essere difficili da determinare.
- Testing metamorfico (vedi [Sezione 3.3.2](#)).
- Oracolo umano: utilizzano la capacità umana di determinare i risultati attesi. Le risorse umane possono essere costose e scarse; tuttavia, sono preferibili in alcuni approcci del test (p.e. testing esplorativo).

Le asserzioni possono essere inserite nel codice di test automation o nell'oggetto di test stesso per implementare un oracolo del test automatizzato. Sono istruzioni eseguibili che verificano lo stato o il comportamento dell'oggetto di test. Quando sono integrate nell'oggetto di test, di norma verificano solo quanto necessario alla continuazione dell'attività.

1.3.5 Requisiti dei Dati di Test

Durante la progettazione dei test, il TA identifica e richiede dati di test che possano necessitare di preparazione o provisioning per l'esecuzione dei test. Le considerazioni includono scopo, formato e contesto d'uso. Anche ([ISO/IEC/IEEE 29119-3](#), 2021, Sezione 8.5) descrive i requisiti dei dati di test. Aspetti chiave sono:

- **Similarità con i dati di produzione.** I dati di produzione riflettono dati del mondo reale ma possono mancare di variabilità. I dati sintetici consentono una variabilità controllata. Dovrebbero riflettere gli aspetti chiave dei modelli, delle distribuzioni e dei valori anomali dei dati di produzione, ma possono trascurare alcuni difetti che si verificano solo con i dati del mondo reale. Per sistemi privi di dati di produzione, i dati sintetici devono riflettere scenari realistici di business e tecnici. Le Persona posso essere utili a fornire profili realistici e user-centered che guidano la creazione di dati rappresentativi

di scenari e comportamenti diversi.

- **Confidenzialità.** I dati di test sensibili (p.e. informazioni personali) richiedono protezione. I dati pseudo-anonimizzati sostituiscono le informazioni personali con identificativi artificiali. I dati anonimizzati rimuovono le informazioni identificabili sugli individui. Se necessario, il TA deve rispettare le normative sulla protezione dei dati come GDPR nell'Unione Europea (Commission, 2016) o HIPAA negli USA (Health et al., 2024).
- **Scopo.** I dati di test sono cruciali per determinare precondizioni e risultati attesi che impattano lo stato del sistema e la sua configurazione (p.e. data e ora del sistema). Includono anche la specifica dei permessi utente e la definizione di relazioni tra prodotti, dipartimenti e categorie.
- **Criteri di copertura.** I dati di test devono allinearsi ai criteri di copertura per la tecnica di test scelta. Oltre ai dati di test validi, possono essere richiesti dati non validi, p.e. per i test negativi.
- **Formato dei dati.** Una gestione sistematica dei dati (p.e. nel testing delle API) può richiedere dati strutturati (p.e. CSV, JSON, XML o database).
- **Tracciabilità.** La tracciabilità assicura la manutenibilità dei dati di test quando si effettuano modifiche ai test case.
- **Manutenibilità.** Evitare dati di test hard-coded nei test case di basso livello per facilitare la rilevazione e manutenzione dei difetti. I test case dovrebbero separare la logica di test dai dati di test (vedi [Sezione 1.3.2](#)).
- **Dipendenze.** Dati dipendenti richiedono una serie di passi per essere creati.
- **Disponibilità.** La virtualizzazione dei servizi può indirizzare il problema dei dati mancanti simulando servizi assenti o inaccessibili per poter interagire con sistemi o servizi esterni.
- **Sensibilità temporale e invecchiamento dei dati.** Test case che utilizzano dati obsoleti o sensibili al tempo possono influenzare il comportamento del sistema in modi inattesi o inaccurati.

1.3.6 Sviluppo di Test Script utilizzando il Testing Keyword-Driven

Se viene utilizzato il testing keyword-driven, il TA crea test script utilizzando le keyword. L'implementazione dei test script è un compito del TTA, del TAE o di uno sviluppatore.

Il TA identifica e specifica le keyword analizzando la base di test o collaborando con gli stakeholder. Le keyword si classificano in due categorie: action keyword e verification keyword. Le action keyword devono interagire con l'oggetto di test (p.e. eseguendo funzioni, sottomettendo dati, navigando all'interno dell'oggetto di test), con l'ambiente di test (p.e. impostando configurazioni e attivando simulatori) o con altri componenti/sistemi (p.e. attivando un'interfaccia dell'oggetto di test). Le verification keyword rappresentano asserzioni per valutare se il risultato effettivo generato dall'oggetto di test corrisponde al risultato atteso.

Le keyword risiedono su almeno due livelli di astrazione: dominio e interfaccia del test. Le keyword a livello di dominio corrispondono ad azioni legate al business e riflettono la terminologia del dominio applicativo, astraggono dai dettagli tecnici dell'interfaccia dell'oggetto di test. Le keyword a livello di interfaccia di test comunicano con gli oggetti di test, con gli elementi dell'ambiente di test o con altri componenti/sistemi tramite la loro interfaccia di test; risiedono sul livello di astrazione più basso. Ulteriori livelli intermedi possono supportare la manutenibilità delle keyword.

La struttura delle keyword può essere atomica o composta da altre keyword. Struttura e livello di astrazione di una keyword sono attributi indipendenti; tuttavia, le keyword composte spesso risiedono a un livello di astrazione più alto, mentre le keyword atomiche tendono a risiedere sul livello di interfaccia di test.

I compiti del TA nel testing keyword-driven includono:

- specificare le keyword e i relativi parametri;
- specificare i test case delle keyword, cioè i test script che usano le keyword;
- specificare i passi aggiuntivi ai test script basati su keyword (p.e. precondizioni, azioni di verifica, pulizia dell'ambiente di test dopo il test);
- mantenere i test case delle keyword per riflettere le modifiche all'oggetto di test;
- eseguire test script basati su keyword, in modo automatizzato o manuale;
- analizzare i test case delle keyword falliti per determinare la causa del failure.

Analizzando la base di test, il TA identifica le interazioni tra oggetto di test e ambiente (p.e. utenti, altri sistemi e dispositivi). Si consideri, ad esempio, la user story “Come socio, voglio autenticarmi, in modo da accedere ai servizi”, con il criterio di accettazione “Le tessere dei soci valide possono essere usate per l'autenticazione”. Il TA può specificare una action keyword “Autenticazione del Socio” con parametro “tessera socio” e una verification keyword “Verifica Accesso”. Il TA verifica che le keyword identificate risiedano sul livello di astrazione appropriato.

Nello specificare le keyword, il TA deve ricordare che le keyword devono:

- contenere un verbo (+ sostantivo);
- utilizzare la forma imperativa del verbo (+ sostantivo);
- essere univoche nel significato;
- essere adeguatamente documentate;
- riflettere il vocabolario del dominio applicativo;
- essere riutilizzabili.

Specificando i test case delle keyword, il TA può rilevare keyword mancanti e specificarle immediatamente. I test case delle keyword possono essere scritti in vari formati, come elenchi o tabelle.

Le keyword possono cambiare nel corso del progetto e sono soggette alla ridondanza (Rwemalika et al., 2019). Le raccomandazioni di questo paragrafo hanno lo scopo di evitare ridondanze e ridurre l'effort di manutenzione.

Benché il testing keyword-driven sia un approccio di test automation, anche il testing manuale può trarne beneficio; applicato al testing manuale, supporta efficacemente una successiva transizione dal manuale verso l'automazione.

Ulteriori dettagli sull'automazione dell'esecuzione dei test e sul testing keyword-driven sono disponibili in (ISTQB CTAL-TTA), (ISTQB CT-TAE) e ([ISO/IEC/IEEE 29119-5](#), 2016).

1.3.7 Strumenti Applicati nella Gestione del Testware

Una corretta gestione del testware attraverso strumenti può aiutare il TA a supportare l'intero processo di test. Gli strumenti possono fornire lo stato dei prodotti di lavoro e supportare il monitoraggio e controllo dei test. In caso di failure nel System Under Test (SUT), consentono al TA di verificare i risultati dei test delle test run precedenti e analizzare il momento in cui si sono verificati i difetti.

Alcune tipologie chiave di strumenti per la gestione del testware includono:

- **Strumenti di test management** per fornire un repository di tutto il testware rilevante (p.e. condizioni di test, test case, test script, test suite e test run). Gli strumenti facilitano la tracciabilità (p.e. tramite la matrice di tracciabilità), il rilevamento dei test case, la pianificazione dei test run, la registrazione dei risultati dei test e il reporting complessivo sull'avanzamento e sulla qualità dei test.

- **Strumenti di defect management** per registrare, prioritizzare e monitorare il processo di risoluzione dei difetti.
- **Strumenti di gestione dei dati di test** per creare e mantenere i dati di test, inclusa la protezione dei dati sensibili (vedi [Sezione 1.3.5](#)).
- **Strumenti di configuration management** per facilitare le attività relative al testing all'interno dei processi di sviluppo, rilascio ed esercizio, inclusa la gestione della configurazione e della disponibilità degli ambienti di test.
- **Strumenti di requirements management** per memorizzare e tracciare i requisiti di alto livello, assicurando che siano chiaramente definiti, tracciati con il versioning e la tracciabilità durante tutto l'SDLC.

Il TA supporta la gestione del testware:

- analizzando il progetto e il rilascio per selezionare il sottoinsieme corretto di testware (p.e. una specifica identificata dalla sua versione per allinearla alla versione del SUT);
- definendo una struttura funzionale (p.e. per funzionalità o moduli) o una struttura tecnica appropriata (p.e. per tipo di test o ambiente di test) per organizzare i test case nello strumento di test management;
- aggiungendo metadati ai test case (p.e. informazioni sull'effort relativo all'esecuzione dei test o allo specifico ambiente di test richiesto);
- assicurando la tracciabilità tra requisiti, condizioni di test, test, test run e difetti;
- selezionando le test suite corrette per il regression testing (per l'esecuzione manuale/automatizzata);
- applicando il configuration management dei test case, inclusa l'identificazione dei test case obsoleti.

Gli strumenti aiutano a organizzare, tracciare e garantire la qualità del processo di test. Il TA supporta questo effort selezionando, strutturando e mantenendo i test case, assicurando la tracciabilità e gestendo le versioni dei test case per un testing ed un controllo dei test efficienti.

2 I Compiti del Test Analyst nel Testing basato sul Rischio – 90 minuti

Parole Chiave

analisi degli impatti, analisi del rischio, controllo del rischio, identificazione del rischio, mitigazione del rischio, monitoraggio del rischio, regression testing, rischio di prodotto, , testing basato sul rischio, valutazione del rischio

Obiettivi di Apprendimento per il Capitolo 2:

2.1 Analisi del Rischio

TA-2.1.1 (K2) Riassumere il contributo del Test Analyst all'analisi del rischio di prodotto

2.2 Controllo del Rischio

TA-2.2.1 (K4) Analizzare l'impatto delle modifiche per determinare l'ambito del regression testing

Introduzione ai Compiti del Test Analyst nel Testing basato sul Rischio

Il testing basato sul rischio è un approccio del test che priorizza le attività di test in base ai livelli di rischio degli elementi di test. Il test manager determina questo approccio. Il TA svolge un ruolo attivo nella sua implementazione. Il testing basato sul rischio è trattato in modo più dettagliato in (ISTQB CTAL-TM).

Il testing basato sul rischio comprende l'analisi del rischio e il controllo del rischio (ISTQB CTFL par. 5.2).

Inoltre, poiché i rischi e i livelli di rischio non sono statici e cambiano nel tempo, il testing basato sul rischio comporta un monitoraggio regolare del rischio. In un ciclo di vita iterativo, la frequenza è stabilita dal team (probabilmente una volta per iterazione). In altri cicli di vita, la frequenza è stabilita dalla figura responsabile della gestione del rischio di prodotto, spesso il test manager. Il TA contribuisce aggiornando il risk register in base alle modifiche apportate e adeguando le azioni di mitigazione del rischio sopra menzionate.

2.1 Analisi del Rischio

L'analisi del rischio comprende l'identificazione e la valutazione del rischio. Questo Syllabus si focalizza su come il TA dovrebbe partecipare alle attività di analisi del rischio per assicurare che il testing basato sul rischio sia implementato correttamente.

Spesso il TA possiede una conoscenza unica del sistema, oltre a esperienza e intuizioni su cosa di solito non funziona, quale impatto ne deriva e come il testing può mitigare il rischio. Questo rende il TA uno stakeholder prezioso per l'analisi del rischio di prodotto.

Nell'identificazione del rischio, il TA contribuisce con la propria esperienza e conoscenza a retrospettive, workshop sui rischi, brainstorming e creazione di checklist. Il TA può anche condurre interviste con gli stakeholder per comprendere meglio quali rischi considerano più significativi dal loro punto di vista.

Durante la **valutazione del rischio**, il TA, insieme agli altri stakeholder, contribuisce a determinare il livello di rischio stimando diversi fattori quali:

- frequenza d'uso e criticità delle funzionalità impattate;
- criticità degli obiettivi di business impattati;
- danno finanziario, ambientale e reputazionale;
- qualità della base di test;
- esigenze legali o di sicurezza.

Il TA aiuta inoltre a classificare i rischi di prodotto in base alle caratteristiche di qualità impattate, ad esempio utilizzando il modello di qualità del prodotto software dello standard [ISO/IEC 25010](#) (2023). Il livello di rischio spesso non è distribuito in modo uniforme nell'oggetto di test. In questi casi, il TA dovrebbe suddividere l'oggetto di test in elementi di test (p.e. componenti, interfacce e funzionalità) e valutare separatamente ogni rischio per un determinato elemento di test.

Infine, come parte della valutazione del rischio di prodotto, il TA propone attività di test adeguate per mitigare ogni rischio identificato. Queste attività possono includere testing statico e testing dinamico. In funzione di fattori come il livello di rischio, il tipo di elemento di test associato e la caratteristica di qualità impattata, il TA indica i livelli di test, i tipi di test, le tecniche di test, i livelli di indipendenza del testing e il grado di profondità della copertura dei test richiesti. In ottica di testing shift-left, il TA indica quali attività di test possano mitigare il rischio il prima possibile, per minimizzare l'effort del testing.

2.2 Controllo del Rischio

Il controllo del rischio comprende la mitigazione del rischio e il monitoraggio del rischio. Il TA conosce le funzionalità del sistema e i rischi potenziali correlati; per questo il ruolo del TA è cruciale in diverse azioni di mitigazione del rischio menzionate in (ISTQB CTFL par. 5.2.4):

- L'esecuzione delle review è trattata nella [Sezione 5.2.2](#) di questo Syllabus.
- L'applicazione delle tecniche di test appropriate e dei relativi livelli di copertura è trattata nella [Sezione 3.5](#).
- L'applicazione dei tipi di test appropriati è trattata nel Capitolo 4.
- L'esecuzione del regression testing è trattata di seguito.

L'obiettivo principale del regression testing è assicurare la fiducia nella qualità dell'oggetto di test dopo una modifica. Tuttavia, durante il regression testing potrebbe non essere possibile eseguire tutti i regression test a causa di vincoli (p.e. limitazioni di tempo, budget, ambiente di test o dati di test). Questo riguarda soprattutto i test eseguiti manualmente, ma si applica anche per i test automatizzati, ad esempio quando i cicli di test sono brevi ed esistono molti test automatizzati che richiedono molto tempo per l'esecuzione. Diventa quindi necessario selezionare i regression test più appropriati in base a criteri specifici. È essenziale eseguire la review l'ambito del regression testing a ogni ciclo di test; il risultato della review può richiedere che la suite di regression test esistente debba essere adeguata.

La tecnica più affidabile per selezionare i test automatizzati è l'analisi degli impatti, che può essere supportata da strumenti specifici. Tali strumenti si basano su un sistema automatizzato di configuration management. Registrano quali configuration item vengono attivati durante l'esecuzione di ciascun test case. Quando avviene una modifica, tracciano quali configuration item sono stati cambiati e selezionano i regression test che interagiscono con gli elementi modificati. In questo modo, gli strumenti assicurano che, dopo una modifica a un configuration item, i test si focalizzino sulle aree dove è più probabile rilevare failure (Juergens et al., 2018).

Per l'esecuzione manuale dei test, non esistono evidenze di una tecnica di selezione dei regression test che sia chiaramente migliore, poiché i risultati dipendono da vari fattori (Engström et al., 2010). Pertanto, il TA deve decidere quale tecnica utilizzare in base alla situazione. Tecniche comunemente impiegate includono:

- **Selezione dei test basata sul rischio**, in cui il TA mantiene la tracciabilità della suite di regression test rispetto al risk register. Quando viene apportata una modifica e il risk register viene aggiornato di conseguenza, il TA adegua la suite di test regression per coprire i livelli di rischio più elevati.
- **Testing basato sulla storia**, in cui il TA valuta le precedenti esecuzioni dei test e determina quali test hanno evidenziato difetti o sono state sensibili a modifiche simili a quella introdotta e che deve essere verificata. Rieseguire ancora questi test come parte del regression testing aumenta la probabilità di rilevare difetti simili. Il TA può includere anche alcuni test non eseguiti da tempo per assicurarsi che vengano ancora eseguiti con successo.
- **Testing basato sulla copertura**, in cui il TA seleziona un numero ridotto di test che permettano di raggiungere la massima copertura possibile in base alle tecniche di test selezionate. La quantità di test deve essere bilanciata con cura in base all'aumento di copertura ottenuto da ogni test.
- **Matrice di tracciabilità dei requisiti**, che il TA utilizza per valutare l'impatto delle modifiche dei

requisiti sui relativi test. È particolarmente utile quando requisiti nuovi o modificati impattano indirettamente funzionalità esistenti. Il TA seleziona i regression test per le funzionalità correlate o direttamente interessate, al fine di coprire possibili effetti collaterali indesiderati. Nello sviluppo software Agile, questo può essere eseguito in modo simile selezionando i test che coprono i criteri di accettazione impattati da user story nuove o modificate.

- **Test basati su profili operativi**, in cui il TA seleziona i regression test da eseguire in base ai modelli di utilizzo dell'oggetto di test. Ad esempio, quando si esegue il testing di un sito e-commerce, uno dei test può includere il login dell'utente, la ricerca dei prodotti, l'inserimento nel carrello e l'invio dell'ordine. Quando un'applicazione viene modificata in modo significativo, questa tecnica fornisce una rapida panoramica della funzionalità complessiva del sistema. Se questo comporta un numero eccessivo di test, il TA priorizza come critici i modelli di utilizzo che si verificano spesso e coprono funzionalità e processi di business critici.
- **L'analisi degli impatti**, che il TA può utilizzare anche per selezionare i regression test da eseguire manualmente se conosce quali interagiscono con i configuration item modificati.

È spesso necessario utilizzare una combinazione di tecniche di selezione dei test per una test suite di regression testing più completa ed efficace. Tuttavia, il TA deve bilanciare attentamente la necessità di copertura con una dimensione gestibile della test suite. Dopo ogni ciclo di test, il TA analizza i risultati dei test per determinare l'efficacia delle tecniche applicate (vedi *Sezione 5.3.1*). Nell'esecuzione successiva, il TA mantiene le tecniche efficaci e sostituisce quelle inefficaci. Questo migliora in modo continuativo la selezione dei regression test nel tempo. È essenziale nei modelli di sviluppo iterativo e incrementale, dove i cambiamenti sono frequenti.

3 Analisi dei Test e Progettazione dei Test – 615 minuti

Parole Chiave

crowd testing, partizione di equivalenza, relazione metamorfica, tecnica di test basata sui dati, tecnica di test basata sul comportamento, test charter, tecnica di test basata sulle regole, testing basato su scenari, testing basato sull'esperienza, testing casuale, testing checklist-based, testing combinatorio, testing CRUD, testing della tabella delle decisioni, testing delle transizioni di stato, testing di dominio, testing metamorfico, testing session-based

Obiettivi di Apprendimento per il Capitolo 3:

3.1 Tecniche di Test basate sui Dati

- TA-3.1.1 (K3) Applicare il testing di dominio
- TA-3.1.2 (K3) Applicare il testing combinatorio
- TA-3.1.3 (K2) Riassumere i vantaggi e i limiti del testing casuale

3.2 Tecniche di Test basate sul Comportamento

- TA-3.2.1 (K2) Spiegare il testing CRUD
- TA-3.2.2 (K3) Applicare il testing delle transizioni di stato
- TA-3.2.3 (K3) Applicare il testing basato su scenari

3.3 Tecniche di Test basate sulle Regole

- TA-3.3.1 (K3) Applicare il testing della tabella delle decisioni
- TA-3.3.2 (K3) Applicare il testing metamorfico

3.4 Testing basato sull'Esperienza

- TA-3.4.1 (K3) Preparare i test charter per il testing session-based
- TA-3.4.2 (K3) Preparare le checklist a supporto del testing basato sull'esperienza
- TA-3.4.3 (K2) Fornire esempi dei vantaggi e dei limiti del crowd testing

3.5 Applicazione delle Tecniche di Test più Appropriate

- TA-3.5.1 (K4) Selezionare tecniche di test appropriate per mitigare i rischi di prodotto in una determinata situazione
- TA-3.5.2 (K2) Spiegare i vantaggi e i rischi dell'automazione della progettazione dei test

Introduzione all'Analisi dei Test e alla Progettazione dei Test

Le tecniche di test sono utilizzate principalmente nell'analisi dei test e nella progettazione dei test. Le tecniche trattate in questo capitolo coprono le tecniche di test black-box e le tecniche di test basate sull'esperienza. Le tecniche di test white-box sono trattate in (ISTQB CTAL-TTA).

Questo Syllabus suddivide le tecniche di test black-box in tre categorie in base alle condizioni di test sottostanti che modellano:

- elementi dei dati (tecniche di test basate sui dati)
- elementi del comportamento dinamico (tecniche di test basate sul comportamento)
- elementi delle regole di comportamento statico (tecniche di test basate sulle regole)

3.1 Tecniche di Test basate sui Dati

In questo paragrafo, il termine “dominio” è utilizzato per rappresentare l'insieme dei dati di input di un elemento di test. Dati di input provenienti da diverse aree del dominio producono comportamenti diversi dell'elemento di test. Le tecniche di test basate sui dati hanno lo scopo di verificare che l'implementazione gestisca correttamente specifiche aree del dominio. Il Syllabus Certified Tester Foundation Level (ISTQB CTFL par. 4.2) copre due tecniche di test classificabili come basate sui dati: il partizionamento di equivalenza (Equivalence Partitioning - EP) e l'analisi ai valori limite (Boundary Value Analysis - BVA). Questo Syllabus introduce tre ulteriori tecniche di test basate sui dati:

- testing di dominio – estende EP e BVA a domini con più parametri e partizioni complesse
- testing combinatorio – si focalizza sulle interazioni di più parametri in un dominio multidimensionale
- testing casuale– seleziona input casuali dal dominio in base a una distribuzione di probabilità specificata

Si noti che il testing casuale può riferirsi in generale sia a dati di input casuali sia a eventi casuali. Questo Syllabus tratta esclusivamente il testing con dati di input casuali.

3.1.1 Testing di Dominio

Il testing di dominio verifica se l'elemento di test si comporta come specificato sulle partizioni di equivalenza del dominio e sui loro confini. In questo caso, le partizioni di equivalenza sono definite utilizzando espressioni che combinano condizioni atomiche mediante operatori booleani (p.e. AND, OR e NOT) e considerano una o più variabili interagenti. Ogni condizione atomica definisce un confine della partizione di equivalenza. I confini chiusi sono formati da espressioni relazionali con operatori $\leq$, $\geq$ o $=$, mentre i confini aperti sono formati da espressioni relazionali con operatori $<$, $>$ o $\neq$.

Ad esempio, l'espressione $\text{altezza} > 1.29 \text{ metri AND peso} / \text{altezza}^2 \geq 30$ definisce una partizione di equivalenza in un dominio bidimensionale (due variabili) con due confini, uno aperto e uno chiuso.

Il testing di dominio cerca di identificare i difetti nelle implementazioni delle partizioni di equivalenza (p.e. operatore o costante errati) selezionando elementi di copertura appropriati in grado di rilevarli. I criteri di copertura nel testing di dominio si riferiscono ai punti ON, OFF, IN e OUT:

- Per un confine chiuso, un punto ON si trova su questo confine. Per un confine aperto, un punto ON si trova all'interno della partizione di equivalenza ed è il più vicino al confine in base alla precisione

data.

- Per un confine chiuso, un punto OFF si trova all'esterno della partizione di equivalenza ed è il più vicino al confine in base alla precisione specificata. Per un confine aperto, un punto OFF si trova su questo confine.
- Un punto IN appartiene alla partizione di equivalenza e non è un punto ON.
- Un punto OUT si trova all'esterno della partizione di equivalenza e non è un punto OFF.

Ciascuno di questi tipi di punti è definito rispetto al confine della partizione di equivalenza; lo stesso punto può assumere tipi diversi rispetto a confini differenti.

I criteri di copertura includono:

Copertura di dominio semplificata (Simplified Domain Coverage; Jeng et al., 1994), che richiede i seguenti elementi di copertura:

- Un punto ON e un punto OFF per ogni confine definito da uno degli operatori $<$, $\leq$, $>$ or $\geq$
- Un punto ON e due punti OFF situati su lati diversi del confine per l'operatore $=$
- Un punto OFF e due punti ON situati su lati diversi del confine per l'operatore $\neq$

Ciascun punto OFF deve essere il più vicino possibile al punto ON corrispondente in base alla precisione specificata.

Copertura affidabile del dominio (Reliable Domain Coverage; Forgács et al., 2024), che richiede i seguenti elementi di copertura:

- Un punto ON, un punto OFF, un punto IN e un punto OUT per ogni confine definito da uno degli operatori $<$, $\leq$, $>$, or $\geq$
- Un punto ON e due punti OFF situati su lati diversi del confine per l'operatore $=$
- Un punto OFF e due punti ON situati su lati diversi del confine per l'operatore $\neq$

Per entrambi i criteri di copertura discussi, il numero di elementi di copertura dipende linearmente dal numero di confini. Può essere ottimizzato per entrambi i criteri di copertura utilizzando lo stesso elemento di copertura per confini diversi. Ad esempio, tutti i confini di una partizione di equivalenza possono utilizzare un punto IN comune oppure una coppia di punti ON e OFF di un confine può essere utilizzata come punti OFF e ON per il confine della partizione di equivalenza adiacente. Per i domini con molti confini, è disponibile un algoritmo di ottimizzazione avanzato (Forgács et al., 2024).

Una copertura di dominio affidabile comporta una quantità leggermente maggiore di elementi di copertura rispetto alla copertura di dominio semplificata, ma è in grado di rilevare un numero notevolmente maggiore di difetti di dominio (*Site of Software Test Design*, 2020).

Il testing di dominio può essere applicato a qualsiasi livello di test. Generalizza BVA ed EP (vedi ISTQBCTFL par. 4.2) a domini più complessi. Vari approcci al testing di dominio possono essere trovati in libri di testo come (Beizer, 1990, cap. 6; Binder, 2000, par. 10.2.4; Kaner; Padmanabhan, et al., 2013; Jorgensen, 2014, cap. 5).

3.1.2 Testing Combinatorio

Alcuni failure software derivano da combinazioni specifiche di valori di parametri, noti come failure di interazione. Il testing combinatorio mira a rivelare tali failure esplorando queste combinazioni di valori di parametri.

Nel testing combinatorio, le condizioni di test combinano generalmente parametri di configurazione o valori di dati di input. Pertanto, esistono due approcci principali al testing combinatorio. Il primo approccio utilizza combinazioni di parametri di configurazione, e ognuna di queste combinazioni può essere testata utilizzando lo stesso test case. Il secondo approccio utilizza combinazioni di valori dei dati di input, che diventano quindi parte di test case completi, generando una test suite per il System Under Test (D. R. Kuhn et al., 2013).

Una coppia parametro-valore specifica è costituita da un parametro e dal suo valore (p.e. "(colore, rosso)"). I criteri di copertura includono (Ammann et al., 2008), (Forgács et al., 2019):

- La **copertura base choice** assume che alcune coppie parametro-valore siano più importanti di altre. Per ogni parametro viene scelta una coppia parametro-valore di base e un elemento di copertura di base è la combinazione delle coppie parametro-valore di base. Gli elementi di copertura successivi vengono creati dall'elemento di copertura di base per ogni parametro, sostituendo la propria coppia parametro-valore di base con ogni valore non di base.
- La **copertura pairwise**, in cui gli elementi di copertura sono coppie (pair) di coppie parametri-valori per due parametri qualsiasi. Esistono strumenti per generare gli elementi di copertura. Tuttavia, l'identificazione di un insieme minimo di test case per raggiungere la copertura pairwise è generalmente difficile.

Se i parametri hanno molti valori, si può applicare prima EP per ridurre il numero di valori e l'insieme delle combinazioni risultanti. Rappresentare i parametri e i loro valori in un albero di classificazione o in un feature model (vedi IREB-Glossario, 2024) supporta questa attività. Il numero finale di test case può essere influenzato da vincoli tra coppie parametro-valore, da combinazioni aggiunte manualmente che sono note essere problematiche, o dalla presenza di combinazioni invalide o non realizzabili.

L'intuizione chiave per il testing combinatorio è che non tutti i parametri contribuiscono a un failure e che la maggior parte dei failure è generato da un singolo valore del parametro o da interazioni tra un numero relativamente piccolo di parametri (Cohen et al., 1994). Questo è in linea con l'ipotesi dell'effetto di accoppiamento, secondo cui il rilevamento di difetti semplici in un programma spesso può rilevare anche difetti complessi (Offutt, 1992). In uno studio limitato (D. Kuhn et al., 2004), i risultati hanno mostrato che circa il 97% dei failure sono causati solo da una o due condizioni interagenti, il che indica che il testing pairwise è una tecnica di test efficace.

Maggiori informazioni sul testing combinatorio, inclusi altri criteri di copertura come il diff-pair-t o il testing n-wise, sono disponibili in (Ammann et al., 2008), (Forgács et al., 2019). Strumenti a supporto del testing combinatorio sono disponibili anche in (Czerwonka, 2004).

3.1.3 Testing Casuale

Il testing casuale consiste nella selezione casuale dei dati di test dal dominio di input dell'elemento di test in base a una distribuzione di probabilità specificata. Ai fini della validazione, si raccomanda una distribuzione basata su profili operativi. Ai fini della verifica, la distribuzione dovrebbe essere indipendente dall'utilizzo per evitare pregiudizi (bias). I risultati attesi potrebbero essere associati ai test case utilizzando un oracolo del test, che più spesso richiede un oracolo del test automatizzato.

Il testing casuale può essere guidato o non guidato. Nel testing casuale non guidato, la distribuzione di probabilità rimane fissa durante tutto il processo. Il testing casuale guidato, esemplificato da tecniche come il testing casuale adattivo (Huang et al., 2019), adatta la distribuzione in base a valori selezionati in precedenza, evolvendo nel tempo. Lo scopo del testing casuale guidato è di coprire in modo efficace il dominio di input, considerando che i difetti spesso si raggruppano in regioni specifiche del dominio.

Il testing casuale non dispone di criteri di copertura riconosciuti. Pertanto, i criteri di uscita possono basarsi solo sul numero di test eseguiti, sulla durata del testing o su misure simili di completamento.

Il testing casuale è particolarmente utile quando la conoscenza del dominio è limitata o quando è necessario un elevato volume di dati di test. È conveniente dal punto di vista economico e fornisce, in termini probabilistici, informazioni dettagliate sull'affidabilità dell'oggetto di test. Il testing casuale aiuta ad evitare pregiudizi come il trascurare difetti nel testing manuale per eccesso di fiducia di alcune parti del codice o di funzionalità. Tuttavia, il testing casuale presenta anche diverse sfide e limitazioni, che includono il trascurare la semantica dei dati, rischiare di non rilevare difetti relativi al significato dei dati, trascurare alcuni difetti, generare test ridondanti, dipendere da un oracolo del test automatizzato e ottenere output casuali che portano a risultati del test inconsistenti. È quindi essenziale bilanciare vantaggi e limiti del testing casuale in ogni contesto specifico del testing.

Tradizionalmente, il testing casuale è considerato meno efficace rispetto ad altre tecniche di test. Negli ultimi anni, questa ipotesi è stata investigata in molti studi empirici. La scoperta è che, nelle circostanze sopra descritte, il testing casuale può essere più efficace ed efficiente rispetto ad altre tecniche di test basate sui dati (Arcuri et al., 2012), (Wu et al., 2020). Il testing casuale viene applicato anche nel fuzz testing e nell'ingegneria del caos (chaos engineering).

3.2 Tecniche di Test basate sul Comportamento

Le tecniche di test basate sul comportamento derivano i test case dalle specifiche del comportamento dinamico (cioè dipendente dallo stato) dell'elemento di test. Questo Syllabus discute tre tecniche di test basate sul comportamento:

- testing CRUD
- testing delle transizioni di stato
- testing basato su scenari

Il testing CRUD e il testing basato su scenari estendono l'insieme delle tecniche di test black-box presenti in (ISTQB CTFL par. 4.2). Questo Syllabus integra il testing delle transizioni di stato con ulteriori criteri di copertura.

3.2.1 Testing CRUD

Il testing CRUD verifica il ciclo di vita delle entità di dati elaborate dall'elemento di test. CRUD significa Create, Read, Update, Delete (creazione, lettura, aggiornamento o eliminazione): sono le quattro operazioni di base che le funzioni possono eseguire sulle entità.

La matrice CRUD offre una panoramica del ciclo di vita delle entità di dati. Le colonne rappresentano le entità, mentre le righe rappresentano le funzioni. Si supponga che una funzione esegua una o più operazioni specifiche di creazione, lettura, aggiornamento o eliminazione su una determinata entità. Questo è indicato nella matrice utilizzando le iniziali di queste operazioni: C, R, U o D. Per creare una matrice CRUD, il TA determina per ogni funzionalità quale delle quattro operazioni esegue su quali entità. Particolare attenzione dovrebbe essere data all'operazione Read (lettura), spesso implicitamente correlata alle operazioni C-U-D.

Il testing CRUD consiste di due parti (Koomen et al., 2006):

- **Il testing di completezza** è un testing statico che verifica se tutte le operazioni possibili (cioè C, R, U e D) si verificano con ogni entità (cioè se l'intero ciclo di vita è stato implementato per ogni entità).

L'assenza di un'operazione è un'anomalia che richiede indagini.

- **Il testing di consistenza** è un testing dinamico che ha lo scopo di integrare le varie funzionalità e verificare se l'entità viene utilizzata in modo consistente. Verifica che le funzioni interagiscano correttamente durante la gestione di un'entità. I test case dovrebbero coprire tutte le operazioni nella matrice CRUD. Inoltre, dovrebbe essere incluso anche il testing negativo (p.e. la lettura di un'entità che non è stata ancora creata). I test case sono progettati per entità, combinando le funzionalità per coprire il suo intero ciclo di vita.

La copertura CRUD è misurata dal rapporto tra il numero di operazioni eseguite dalla test suite e il numero totale di operazioni nella matrice CRUD. Una copertura CRUD più rigorosa può considerare combinazioni specifiche di operazioni come elementi di copertura (p.e. dopo ogni U, tutte le R possibili dovrebbero essere coperte).

Il testing CRUD è utilizzato principalmente a livello di sistema. Si focalizza sui difetti nel comportamento dell'elemento di test nella gestione delle entità, come violazioni dell'integrità dei dati, difetti di controllo degli accessi o incongruenze dei dati.

3.2.2 Testing delle Transizioni di Stato

Molti sistemi complessi sono stateful (cioè la reazione del sistema a un evento dipende dallo stato attuale del sistema). Esempi di sistemi stateful sono i sistemi embedded, i sistemi basati su dialoghi, i sistemi di controllo o i sistemi che trattano entità e i relativi cicli di vita.

Uno stato semplifica dettagli interni complessi, rendendo più facile per gli stakeholder comprendere il comportamento previsto. Durante l'analisi dei test, il TA deve garantire che il modello basato sugli stati rappresenti il comportamento atteso dell'elemento di test con il livello di dettaglio necessario per il testing. Se la base di test contiene già un modello di questo tipo, il TA deve verificare se contiene le condizioni di test e, se necessario, adattarlo, oppure progettare un nuovo modello.

Nei modelli basati sugli stati, i nodi rappresentano gli stati e le linee rappresentano le transizioni di stato. Le transizioni di stato sono attivate da eventi e possono includere guard condition e azioni (vedi ISTQB CTFL par. 4.2.4). Esistono diverse varianti di questi modelli, come le macchine a stati finiti estese (Bochmann et al., 1994), gli state chart di Harel (Harel, 1987) o le state machine UML (OMG® UML, 2017).

Oltre ai criteri di copertura delle transizioni di stato discussi in (ISTQB CTFL), di seguito vengono discussi altri due criteri per i quali è stata empiricamente dimostrata un'elevata efficacia nel rilevamento dei difetti:

- **La copertura N-switch** si applica a sequenze valide di N+1 transizioni consecutive, chiamate anche N-switch (Chow, 1978). La copertura 0-switch è uguale alla *copertura delle transizioni valide* (vedi ISTQB CTFL par. 4.2.4). Uno 1-switch è una coppia di transizioni in entrata e in uscita in uno stato. L'estensione di un N-switch a partire dallo stato terminale con una transizione di stato valida permette di ottenere un N+1 switch. Generalmente sono utilizzate nella pratica le coperture 0-switch e 1-switch. Una copertura 2-switch o superiore pari al 100% è richiesta solo in caso di alto rischio di failure dovuto a sequenze di eventi non attese, poiché il numero di N-switch può crescere in modo esponenziale con N.
- **La copertura round-trip** (Ammann et al., 2008) si applica ai percorsi in un modello basato sugli stati che formano dei loop. Un round trip è un loop in cui gli stati iniziale e finale sono gli stessi e nessun altro stato in questo loop si verifica due volte. Gli elementi di copertura sono i round trip. (Antoniol et al., 2002) hanno scoperto che questo criterio è molto efficace nel rilevare i difetti.

Il testing delle transizioni di stato è particolarmente adatto all'automazione attraverso strumenti di testing

model-based. Come la maggior parte delle tecniche di test black-box, il testing delle transizioni di stato contribuisce alla prevenzione dei difetti rilevando i difetti nelle specifiche durante la modellazione. Il testing delle transizioni di stato può essere applicato a qualsiasi livello di test.

Ulteriori criteri di copertura delle transizioni di stato sono discussi in (Rechtberger et al., 2022). Un modello basato sugli stati recente è il modello azione-stato discusso in (Forgács et al., 2024).

3.2.3 Testing basato su Scenari

Il testing basato su scenari valuta il comportamento dell'elemento di test in scenari realistici. Tecniche quali la user research, le user story, le Persona e le user journey map (vedi par. 11) possono aiutare a identificare scenari significativi. Nel testing basato su scenari, il TA crea un modello di scenario basato su sequenze di azioni che costituiscono i workflow attraverso l'elemento di test (cfr. ISO/IEC/IEEE 29119-4, 2021). Questo Syllabus descrive i seguenti due modelli: activity diagram e use case. Altri modelli includono flowchart, diagrammi Business Process Model and Notation (OMG® BPMN, 2013), sequence diagram o collaboration diagram (OMG® UML, 2017). Questi modelli non sono trattati nel presente Syllabus. Per ulteriori dettagli, fare riferimento a (ISTQB CT- AcT).

Un **activity diagram** è una rappresentazione grafica del workflow all'interno di un sistema. Gli activity diagram sono particolarmente utili per modellare i processi di business, ma possono anche modellare control flow. Gli activity diagram estendono la notazione dei flowchart, consentendo la modellazione della concorrenza. Gli elementi principali degli activity diagram sono i nodi di inizio e di fine, le azioni, le transizioni, i nodi decisionali, i nodi merge, i nodi fork, i nodi join e le swim lane.

Uno **use case** è una descrizione testuale o grafica delle azioni che rappresentano le interazioni tra un utente e un sistema o tra sistemi. Nel modello si distinguono tre tipi di scenari:

- **Scenario principale** (il cosiddetto "happy path") – una sequenza tipica e attesa di azioni che portano al raggiungimento di un obiettivo specifico dal punto di vista dell'utente. Uno use case può avere un solo scenario principale.
- **Estensione** (o scenario alternativo): una sequenza di azioni, diversa dallo scenario principale, che alla fine porta al raggiungimento dell'obiettivo dello scenario principale.
- **Eccezione**: una sequenza di azioni che non consente il raggiungimento dell'obiettivo dello scenario principale a causa di un'azione non attesa (p.e. un uso anomalo o un input invalido).

Nel testing basato su scenari, il TA progetta i test case per coprire gli scenari (cioè gli elementi di copertura), spesso seguendo una prioritizzazione basata sul rischio. Quando il modello dello scenario non contiene loop (cicli), ogni scenario può essere testato con un test case separato (ovvero tutti gli scenari possibili nel modello possono essere esercitati con una test suite). Il numero di scenari potenziali (cammini) può essere infinito quando esistono loop. In questo caso, si può applicare la copertura del loop semplice. La copertura del loop semplice richiede il testing quando ogni loop viene eseguito zero volte (cioè saltato), con esattamente un'iterazione, con più di un'iterazione (cioè un numero tipico di iterazioni) e con il numero massimo di iterazioni (se possibile).

La **copertura basata su scenari** è misurata dal rapporto tra numero di scenari eseguiti e numero di tutti gli scenari identificati. Gli scenari possono essere identificati applicando diversi criteri di copertura al modello dello scenario (vedi Koomen et al., 2006). I criteri di copertura possono richiedere di testare lo stesso scenario più volte. Ad esempio, uno scenario può richiedere una copertura EP o BVA aggiuntiva delle variabili che si verificano nello scenario. In questi casi, uno scenario può dover essere verificato con più di un test case.

Il testing basato su scenari viene spesso eseguito durante il testing di sistema o il testing di accettazione.

Assume la forma di testing end-to-end focalizzato sull'adeguatezza funzionale di un sistema (vedi par. 4.1) dal punto di vista dell'utente. Tuttavia, il testing basato su scenari può essere utilizzato anche ad altri livelli di test (p.e. testing di integrazione dei componenti con scenari basati sui protocolli di interazione o testing di componente di classi object-oriented e stateful con scenari che invocano vari metodi) e nel testing non-funzionale (p.e. gli scenari possono costituire gli elementi dei profili operativi utilizzati nel testing di affidabilità, di flessibilità o di compatibilità).

3.3 Tecniche di Test basate su Regole

Le tecniche di test basate su regole verificano l'implementazione del comportamento di un elemento di test, specificato da regole valide indipendentemente dal suo stato (p.e. regole di business). In questo Syllabus vengono discusse due tecniche di test basate su regole:

- testing della tabella delle decisioni
- testing metamorfico

3.3.1 Testing della Tabella delle Decisioni

Il Syllabus (ISTQB CTFL) copre i fondamenti del testing della tabella delle decisioni. Questo Syllabus si basa su questo e tratta argomenti più avanzati. La terminologia e la notazione utilizzate seguono lo standard (OMG® DMN, 2024).

Nel testing della tabella delle decisioni, il TA inizia generalmente creando una tabella delle decisioni completa o analizzando una tabella esistente ottenuta dalla base di test. Il numero di regole di una tabella delle decisioni completa è il prodotto del numero di valori delle sue condizioni. Questo numero cresce in modo esponenziale con il numero delle condizioni e dei loro valori, e motiva la minimizzazione delle tabelle delle decisioni. La minimizzazione parte da una tabella delle decisioni originale, che può essere completa o meno, e deriva una tabella delle decisioni equivalente con meno regole.

Le tabelle delle decisioni possono essere minimizzate unendo le regole utilizzando l'operatore "—" ("don't care"). Quando si uniscono le regole, si raccomanda di ignorare le regole non fattibili (cioè le regole con una combinazione di valori di condizioni che non possono mai verificarsi). Spesso, le regole non fattibili vengono eliminate dalla tabella delle decisioni prima dell'unione. Un possibile algoritmo di minimizzazione sistematica cerca regole equivalenti in termini di azione che differiscono solo per una condizione e coprono tutti i suoi valori possibili; tali regole vengono unite sostituendo con "—" il valore della condizione che differisce. Il risultato della minimizzazione sistematica può dipendere dall'ordine in cui le colonne vengono minimizzate, non sempre portando a una soluzione ottimale. Il TA deve verificare se è possibile un'ulteriore minimizzazione.

È compito del TA eseguire la review della tabella delle decisioni, possibilmente con altri stakeholder, in base ai seguenti criteri:

- consistenza (cioè se due regole diverse si applicano alla stessa combinazione di valori delle condizioni, allora sono equivalenti in termini di azione)
- fattibilità (cioè non contiene regole non fattibili)
- completezza (cioè nessuna combinazione fattibile di valori delle condizioni è mancante)
- correttezza (cioè le regole modellano il comportamento atteso del sistema)

Inoltre, è consigliabile che le regole non si sovrappongano (cioè per qualsiasi combinazione di valori delle condizioni, si applica al più una regola). Le regole sovrapposte possono verificarsi quando la tabella delle

decisioni originale è già stata minimizzata o se le regole sono state unite in modo errato.

La *procedura di checksum* utilizza il numero di regole in una tabella delle decisioni minimizzata per indicare sovrapposizioni e lacune. Per ogni regola nella tabella minimizzata, viene calcolato il numero di regole che essa rappresenta nella tabella delle decisioni originale. Una regola senza un "-" nelle condizioni (cioè con valori singoli per tutte le condizioni) ottiene un punteggio di 1. Ogni valore "-" moltiplica il punteggio della regola per il numero dei singoli valori della rispettiva condizione. La somma dei punteggi delle regole è il checksum della tabella delle decisioni minimizzata. Se il checksum è inferiore al checksum della tabella delle decisioni originale, la tabella delle decisioni minimizzata è incompleta. Se il checksum è superiore, alcune regole si sovrappongono o esistono regole aggiuntive (p.e. combinazioni di condizioni non fattibili). Se la minimizzazione è stata eseguita correttamente, i checksum sono uguali. La sola uguaglianza dei checksum non garantisce che la tabella delle decisioni minimizzata sia equivalente a quella originale.

La **copertura della tabella delle decisioni** viene misurata dal rapporto tra il numero di colonne esercitate e il numero totale di colonne fattibili nella tabella delle decisioni. Nell'implementare i test case risultanti da una regola della tabella delle decisioni, il TA deve implementare le condizioni e le azioni. Il TA deve decidere come implementare i valori della condizione "-" per una data regola, poiché "-" rappresenta almeno due valori. Tuttavia, se il livello di rischio associato alla tabella delle decisioni è alto, il TA dovrebbe evitare la minimizzazione e dovrebbe misurare la copertura delle colonne fattibili nella tabella delle decisioni completa.

3.3.2 Testing Metamorfico

Il testing metamorfico (MT) è una tecnica che ha lo scopo di generare i test case basandosi su un test case sorgente esistente. Uno o più test case di follow-up vengono generati modificando (metamorfizzando) il test case sorgente sulla base di una relazione metamorfica (MR). La relazione metamorfica definisce una proprietà dell'elemento di test e descrive come una modifica degli input di un test case si riflette nei risultati attesi del test case.

Il TA combina i test case sorgente e i test case di follow-up di una relazione metamorfica in una test procedure, valutando congiuntamente i risultati: se la relazione metamorfica è soddisfatta, il test è superato; altrimenti risulta fallito. In caso di failure, il successivo debugging deve stabilire quale dei singoli test case interessati è fallito.

Si consideri, ad esempio, una funzione che calcola la media di una serie di numeri. Viene creato un test case sorgente con una serie di numeri e una media attesa, e il test case viene eseguito per confermare che passa. Una relazione metamorfica potrebbe affermare che qualsiasi permutazione della serie di numeri genera come risultato la stessa media. Utilizzando questa relazione metamorfica, il TA può creare diversi test case di follow-up, ognuno con lo stesso insieme di numeri in input ma in un ordine diverso. Il risultato atteso rimane lo stesso.

Per la stessa funzione che calcola la media, il TA può anche utilizzare un'altra relazione metamorfica che afferma che, se ogni numero della serie viene moltiplicato per lo stesso numero x , allora anche il risultato atteso sarà moltiplicato per x . Utilizzando questa relazione metamorfica, il TA può creare un numero qualsiasi di test case di follow-up da un test case sorgente, selezionando valori differenti di x . Questo può essere particolarmente utile quando la progettazione dei test e l'esecuzione dei test sono automatizzate. Il TA può anche combinare due o più MR per creare test case di follow-up (p.e. permutare e moltiplicare per 2).

Il testing metamorfico può essere utilizzato anche in presenza di un problema di oracolo del test (vedere il par. 1.3.4). In tale situazione, non sono disponibili i risultati attesi del test case sorgente e dei test case di follow-up, quindi i risultati dei test non possono essere valutati singolarmente. Un esempio può essere

un programma attuariale basato sull'intelligenza artificiale che prevede l'età alla morte sulla base di un ampio set di dati. Una relazione metamorfica potrebbe affermare che se il numero di sigarette fumate aumenta, l'età prevista di morte dovrebbe diminuire.

Attualmente non esistono misure di copertura riconosciute per il testing metamorfico che forniscano criteri di uscita utili. Coprire ogni relazione metamorfica una sola volta non è sufficiente perché è possibile ottenere solo una verifica parziale dei risultati attesi. Il TA può combinare il testing metamorfico con il testing casuale per generare molti test case sorgente di basso livello e test case di follow-up per la stessa relazione metamorfica. Ad esempio, nel calcolo della media sopra menzionato, è possibile utilizzare un generatore di numeri casuali per generare vari input per il test case sorgente, nonché permutazioni e moltiplicatori casuali per i test case di follow-up.

Il testing metamorfico può essere utilizzato per la maggior parte degli elementi di test e applicato al testing funzionale e non-funzionale (p.e. il testing di carico, generando il carico attraverso test case di follow-up grazie alle relazioni metamorfiche, o il testing di installabilità con diversi parametri di installazione che possono essere selezionati in sequenze multiple). È una tecnica di test ritenuta molto efficace e preferibile per i sistemi basati sull'intelligenza artificiale (ISTQB CT-AI).

Per ulteriori dettagli, vedi anche lo standard (*ISO/IEC/IEEE 29119-4*, 2021) o gli articoli di approfondimento (Segura; Towey, et al., 2020) e (Segura; Fraser, et al., 2016).

3.4 Testing basato sull'Esperienza

Un TA impiega approcci e tecniche di test basati sull'esperienza, sfruttando la propria competenza e le esperienze passate per guidare il testing. Questo paragrafo descrive in dettaglio l'uso della documentazione da parte del TA nel testing session-based e nel testing checklist-based (vedi ISTQB CTFL par. 4.4.2 e 4.4.3). Inoltre, viene discusso il crowd testing. Tutte le tecniche di test basate sull'esperienza possono essere applicate nel testing basato sulla collaborazione, come l'Acceptance Test-Driven Development. Per maggiori dettagli, si veda (ISTQB CTFL par. 4.5).

3.4.1 Test Charter a supporto del Testing Session-based

Nel testing esplorativo, un test charter fornisce la missione di una sessione di test, delineandone l'ambito, gli obiettivi e informazioni quali limitazioni, tempistiche e rischi. Funge da roadmap per il testing, fornendo una struttura alle sessioni di test. I test charter aiutano il TA a rimanere focalizzato su aree o funzionalità specifiche da testare, consentendogli al contempo la flessibilità necessaria per esplorare il sistema quando necessario. Il test charter non specifica le test suite che saranno eseguite in ciascuna sessione di test.

Quando si prepara un test charter per il testing session-based, il TA deve considerare alcuni fattori che influenzano la progettazione del test charter, in particolare:

- fattori relativi al cliente e ai requisiti (p.e. requisiti elicitati dai clienti, use case di business per il sistema, requisiti di qualità) e user journey map (cioè l'interazione dell'utente con il sistema nel tempo)
- fattori relativi al prodotto (p.e. flussi funzionali, obiettivi principali del prodotto, funzionalità del prodotto, progettazione del software e interfacce)
- fattori relativi al project management (p.e. vincoli di tempo, scopo del progetto, effort stimato e valore di business)

Un test charter consiste di una missione che descrive l'obiettivo del test e varie informazioni aggiuntive.

Un formato leggero (lightweight) di una missione è "Esplora [obiettivo] con [risorse] per scoprire [informazioni]"

(Hendrickson, 2013), dove [obiettivo] descrive ciò che deve essere esplorato (p.e. area, funzionalità, rischio, componente e requisito), [risorse] descrive quello che il TA utilizzerà (p.e. dati di test, configurazioni, strumenti, restrizioni, euristiche e dipendenze) e [informazioni] spiega quale tipo di informazioni il TA intende trovare (p.e. valutazioni delle caratteristiche di qualità, tipo di difetti attesi e violazioni degli standard).

I test charter possono contenere, ma non sono limitati a, le seguenti informazioni aggiuntive (Ghazi et al., 2017):

- informazioni organizzative (p.e. durata della sessione di test, data e ora di inizio e nome del tester)
- obiettivi del test (p.e. motivazione del test e missione del test charter)
- ambito del test (p.e. aree specifiche di interesse all'interno del System Under Test, livello di test, tecniche di test da utilizzare, idee di test, criteri di uscita, priorità, cosa dovrebbe coprire il test charter e una descrizione di quello che non sarà testato)
- criteri di ingresso (cioè le precondizioni che devono essere soddisfatte per poter avviare la sessione di test)
- informazioni relative al prodotto (p.e. definizione, dati e workflow tra i componenti e architettura del sistema)
- limitazioni (cioè quello che il prodotto non deve mai fare)
- descrizione dell'ambiente di test
- sorgenti dei dati esistenti, informazioni sul prodotto e strumenti di test che potrebbero aiutare il testing
- informazioni storiche (p.e. difetti rilevati in precedenza, come i difetti di compatibilità e interoperabilità, punti aperti relativi ad anomalie esistenti e modelli di failure relativi ai test effettuati in passato)
- vincoli e rischi (p.e. normative, regole e standard utilizzati)

Durante una sessione di test, il TA registra i test log che includono domande, osservazioni o idee per il testing futuro insieme ai risultati del test. Questi dati sono documentati in un session sheet.

L'ambito e il dettaglio delle informazioni incluse in specifici test charter possono variare e influire sul grado di flessibilità del TA. Ad esempio, definire solo gli obiettivi generali del test offre ampio spazio per l'esplorazione, mentre aggiungere informazioni sulle tecniche di test da utilizzare può limitare il TA. D'altra parte, l'aggiunta di informazioni (p.e. quello che il sistema non può assolutamente fare) può contribuire a ridurre la probabilità di segnalare risultati falsi-positivi.

3.4.2 Checklist a supporto delle Tecniche di Test basate sull'Esperienza

Il testing checklist-based è una tecnica di test ampiamente utilizzata grazie alla sua adattabilità, semplicità ed efficacia nel garantire la qualità del software. Utilizzando le checklist, il TA assicura di coprire tutti gli aspetti essenziali noti di un elemento di test, evitando di trascurare aree critiche. Inoltre, le checklist introducono consistenza tra i cicli di test e tra i diversi TA. Quando usa una checklist, il TA si focalizza sugli aspetti importanti. Le checklist sono una forma di registrazione delle esperienze del TA rispetto a failure e difetti del passato. Servono come promemoria o fonte di ispirazione se il TA esaurisce le idee (p.e. durante il testing esplorativo). Il TA risparmia tempo riutilizzando una checklist standard o una

checklist creata da uno dei suoi colleghi, ma solo se queste checklist sono rilevanti per l'elemento di test. Le checklist aiutano a ridurre il lavoro necessario per documentare i test case, che questo può essere una grande risorsa quando i requisiti e il software sono continuamente modificati.

La preparazione delle checklist è spesso sotto la responsabilità del TA. Le checklist supportano il testing basato sull'esperienza, poiché aiutano a organizzare, strutturare e guidare il testing. Creare una checklist riutilizzabile, manutenibile, chiara ed efficiente richiede impegno.

I seguenti passi possono essere una linea guida per impostare una checklist adeguata al testing basato sull'esperienza:

In primo luogo, il TA determina l'ambito, gli obiettivi e il formato della checklist, poiché questi fattori influenzano la profondità del testing e il livello di dettaglio richiesto. Una checklist read-do contiene gli elementi principali da considerare per un determinato processo (p.e. gli elementi della checklist contengono input invalidi concreti per un campo di input da verificare). Una checklist do-confirm serve come aiuto per guidare il processo di ragionamento, fornendo idee per il testing basate sull'esperienza in modo da esplorare ulteriormente un'applicazione (p.e. un elemento della checklist potrebbe richiedere di verificare se i risultati della ricerca sono pertinenti).

Successivamente, il TA raccoglie le informazioni necessarie per definire gli elementi della checklist. Questo può includere la raccolta di informazioni da professionisti esperti, la consultazione di librerie dei difetti e tassonomie dei difetti (Beizer, 1990), (Kaner; Falk, et al., 1999), la review della documentazione rilevante, l'analisi dei rischi, dei test case e dei potenziali scenari. Gli elementi della checklist dovrebbero essere chiari, specifici, non ambigui, consistenti, rilevanti, manutenibili, fattibili e misurabili. Dovrebbero essere formulati come domande a cui è possibile rispondere con "sì", "no" o "non applicabile". Richiedono di essere prioritizzati in base alla loro importanza, al potenziale impatto e al livello di rischio. Le checklist non sono guide complete. Sono invece strumenti rapidi e semplici per professionisti esperti.

Infine, il TA struttura e organizza la checklist classificando gli elementi della checklist in gruppi logici basati su aree funzionali, ruoli dell'utente, livelli di test o altri criteri rilevanti. La creazione di categorie separate può essere particolarmente utile per una checklist estesa.

Ove possibile, dovrebbero essere utilizzati template e standard. Il TA può risparmiare tempo utilizzando template esistenti o checklist predefinite in linea con gli standard di settore e le best practice.

Una checklist non è mai definitiva. Il TA la rivede e la raffina continuamente, adattandola per riflettere nuove scoperte, cambiamenti nelle priorità, feedback da altri tester o lessons learned dai cicli di test precedenti. Condividendo la checklist con altri tester, il TA promuove la consistenza e la collaborazione, e li aiuta a comprendere meglio gli elementi di test e le aree critiche su cui focalizzarsi durante il testing.

3.4.3 Crowd Testing

Il crowd testing distribuisce i test tra un gruppo di tester interni o esterni con background e ubicazioni diverse. Può essere un modo economico per validare l'usabilità e copre le caratteristiche di qualità sia funzionali sia non-funzionali (Alyahya, 2020), (Leicht et al., 2017).

Alcuni dei vantaggi del crowd testing includono:

- **Ambienti di test diversificati.** I tester possono trovarsi in varie aree geografiche e utilizzare diverse configurazioni dell'ambiente con un'ampia varietà di dispositivi, browser e condizioni di rete.
- **Maggiore flessibilità.** Il crowd testing è facilmente scalabile per gestire molti test in breve tempo.
- **Efficacia nei costi.** Il crowd testing è in genere meno costoso rispetto al mantenimento di un team di test in-house ampio e diversificato o rispetto all'integrazione con servizi di testing esterni.

- **Feedback rapido.** I tester possono fornire un feedback tempestivo, contribuendo a identificare e correggere in anticipo i failure.
- **Prospettiva dell'utente reale.** I tester possono essere utenti reali dell'applicazione e possono fornire informazioni più approfondite sulla propria user experience e usabilità. Questo può essere particolarmente utile durante lo user acceptance testing.
- **Variabilità.** Poiché i test vengono eseguiti ogni volta da un'ampia gamma di tester, non sono ripetibili. Sebbene questo potrebbe essere una limitazione, genera anche una copertura più ampia, che aumenta le possibilità di rilevare i difetti.

Alcuni dei limiti del crowd testing includono:

- **Qualità del testing non affidabile.** La qualità del testing può variare in modo significativo a seconda delle competenze dei singoli tester, anche se questo può non essere rilevante, ad esempio, quando l'obiettivo è ottenere un feedback sulla user experience.
- **Sfide di comunicazione.** Il coordinamento con molti tester distribuiti geograficamente con fusi orari, differenze culturali e barriere linguistiche differenti può essere difficile.
- **Sicurezza.** La condivisione di software con tester esterni comporta rischi per la sicurezza e la riservatezza dei dati. Misure appropriate possono mitigare questi rischi, consentendo un uso responsabile del crowd testing senza divulgare dettagli sensibili o facilitare il plagio.
- **Documentazione e reportistica.** Garantire una documentazione completa del test e gestire un gran numero di segnalazioni, inclusi duplicati e falsi positivi, può essere sfidante quando si ha a che fare con un gruppo ampio ed eterogeneo di tester.

Il crowd testing è un approccio che non sostituisce i TA che applicano tecniche di test, ma può aumentare la copertura di diversi ambienti di test.

3.5 Applicazione delle Tecniche di Test più Appropriate

Il testing dovrebbe essere il più efficace ed efficiente possibile nel contesto dato. A tal fine, il TA supporta il test manager nella selezione delle tecniche di test più appropriate. Inoltre, il TA può utilizzare l'automazione per supportare le attività di test. Questo include l'automazione della progettazione dei test, descritta in questo paragrafo, e il supporto all'automazione dell'esecuzione dei test, descritta nel par. 1.3.6.

3.5.1 Selezione delle Tecniche di Test per Mitigare i Rischi di Prodotto

Selezionare le tecniche di test più appropriate è fondamentale per una mitigazione efficace ed efficiente del rischio di prodotto, ed è influenzata da molti fattori, tra cui i seguenti.

Gli **obiettivi del test** (vedi ISTQB CTFL par. 1.1.1) specificano quali aspetti dell'oggetto di test devono essere valutati. Guidano la scelta delle tecniche di test e dipendono dal tipo di sistema (p.e. testing basato sul dominio per calcoli numerici nell'ingegneria rispetto al testing della tabella delle decisioni nella gestione del rischio relativo al credito).

I **rischi di prodotto** sono associati ai potenziali difetti. Questi difetti possono essere meglio rilevati utilizzando particolari tecniche di test, poiché la maggior parte delle tecniche di test si focalizza sul rilevamento di specifici tipi di difetti (vedi i par. 3.1, 3.2, 3.3 e 3.4 del presente Syllabus). Ad esempio:

- Le tecniche di test basate sui dati possono rilevare difetti nella gestione dei dati, nell'implementazione del dominio, nelle interfacce utente, nei calcoli e nelle combinazioni di parametri.

- Le tecniche di test basate sul comportamento possono rilevare difetti dei requisiti dell'utente, come funzionalità mancanti, difetti di comunicazione e difetti di elaborazione.
- Le tecniche di test basate sulle regole possono rilevare difetti nella logica e nei flussi di controllo.

L'analisi del rischio aiuta anche a determinare l'approccio del test appropriato, ad esempio:

- Criteri di uscita basati sulla copertura. Più alto è il livello di rischio, più rigorosa può essere la copertura richiesta (p.e. tutte le combinazioni nella copertura combinatoria invece della copertura pairwise). Tuttavia, il TA deve sempre tenere conto del compromesso tra la forza della copertura e l'effort del test richiesto.
- Il testing basato sull'esperienza può essere utilizzato quando è difficile definire la copertura, quando il livello di rischio è basso o quando la schedulazione del progetto è stretta.

Base di test. Se la specifica dell'oggetto di test utilizza modelli, il TA può applicare tecniche di test basate su questi modelli. Se è impossibile o difficile derivare un oracolo del test dalla base di test, è possibile applicare tecniche come il testing metamorfico o tecniche di test basate sull'esperienza.

La conoscenza dei tipi di difetti ricorrenti può orientare la scelta delle tecniche di test che si focalizzano sul rilevamento di questi difetti (p.e. testing checklist-based). Se ci si aspetta di rilevare difetti simili a quelli rilevati in iterazioni o progetti precedenti, può essere ragionevole utilizzare le stesse tecniche di test risultate efficaci in precedenza.

Conoscenza ed esperienza del tester. Se il TA non ha familiarità con una determinata tecnica di test, non è consigliabile utilizzarla per identificare test critici. Anche la conoscenza del dominio può influire sulla selezione delle tecniche di test. Ad esempio, una conoscenza del dominio scarsa o assente indica che tecniche come il testing esplorativo possono essere inefficaci.

Ciclo di vita dello sviluppo software utilizzato. Un modello di sviluppo sequenziale favorisce l'impiego di tecniche più formali. Al contrario, un modello di sviluppo iterativo potrebbe essere più appropriato per l'adozione di tecniche di test leggere (p.e. tecniche di test basate sull'esperienza) o quando la progettazione dei test può essere automatizzata.

Requisiti del cliente e del contratto. I contratti possono richiedere esplicitamente l'esecuzione di un testing specifico (p.e. in termini di determinati livelli di test o tipi di test), che influenza la selezione delle tecniche di test (p.e. i criteri di accettazione con una serie di scenari forniti dal cliente suggeriscono l'uso di una tecnica di test basata su scenari).

Requisiti normativi. Quando un progetto segue uno standard, può richiedere l'uso di tecniche di test specifiche. Ad esempio, lo standard (ISO 26262, 2018) richiede l'uso di tecniche di test quali il partizionamento di equivalenza, l'analisi ai valori limite o l'error guessing, in base all'ASIL (Automotive Safety Integrity Level) assegnato a un oggetto di test.

Vincoli di progetto. Vincoli come il tempo e il budget, possono influire sull'uso di tecniche che richiedono molto tempo o risorse costose.

Le tecniche di test vengono spesso combinate per aumentare l'efficienza e l'efficacia del rilevamento dei difetti. Ad esempio:

- Il BVA può essere utilizzato per le guard condition nel testing delle transizioni di stato.
- Il testing di dominio può essere utilizzato nel testing basato su scenari per determinare il valore di una condizione a partire da una tabella delle decisioni, o il valore di una variabile presente in un System Under Test.

- Il testing basato su scenari può essere combinato con la copertura delle decisioni, una tecnica di test white-box descritta in (ISTQB CTAL-TTA), per coprire rigorosamente le decisioni nel processo di business. Per un esempio, si veda il testing del ciclo di processo in (Koomen et al., 2006).
- Il testing basato su scenari può essere combinato con la copertura round-trip per indirizzare rischi specifici delle attività di un processo di business ciclico.

3.5.2 Vantaggi e Rischi dell'Automazione della Progettazione dei Test

Il TA può utilizzare strumenti per applicare tecniche di test, in particolare tecniche di test black-box. Quando la progettazione dei test viene automatizzata, il TA crea un modello di test e genera automaticamente il testware da tale modello. Ad esempio, il TA progetta uno state transition model lascia che uno strumento di test model-based generi test case per la copertura round-trip.

L'automazione della progettazione dei test spesso migliora l'efficienza e l'efficacia del testing. I benefici includono:

- **Prevenzione dei difetti.** La modellazione per il testing è un modo efficace per valutare la qualità della base di test (vedi par. 5.2.1).
- **Maggiore potenzialità.** L'automazione consente l'applicazione di tecniche di test e criteri di copertura più complessi quali il testing combinatorio, il testing casuale o la copertura N-switch, riducendo il rischio di codice non testato.
- **Miglioramento della comprensibilità.** I criteri di selezione dei test specificati in uno strumento fanno riferimento in modo più chiaro e visibile alle condizioni di test, e giustificano in modo più comprensibile la copertura generata.
- **Meno lavoro ripetitivo.** Il testware può essere generato dal modello di test, riducendo il lavoro manuale e ripetitivo come la progettazione dei test.
- **Meno effort di manutenzione.** Il modello di test è l'unica sorgente di verità utilizzata per derivare il testware. Di conseguenza, deve essere mantenuto solo il modello di test.
- **Meno testware difettoso.** Il lavoro manuale è soggetto a errori. Il testware generato dagli strumenti ha una qualità e una consistenza più alta.
- **Migliore collaborazione del team.** Gli stakeholder possono eseguire la review del modello di test per rilevare difetti o comprendere meglio le condizioni di test.
- **Tracciabilità potenziata.** È più facile collegare gli elementi di un modello di test alle condizioni di test piuttosto che ai test case stessi. Se supportati dallo strumento, i test case generati ereditano tali collegamenti, migliorando la tracciabilità complessiva nel testing.
- **Vari formati di output.** Il testware può essere generato in vari formati di output, in base ai requisiti di altri strumenti e delle attività successive.

Il TA deve anche considerare i rischi dell'automazione della progettazione dei test. Questi includono: trascurare le condizioni di test che non sono evidenziate nel modello, sottostimare l'effort di manutenzione del modello di test, avere stakeholder che hanno difficoltà a comprendere il modello, e i rischi generici della test automation (vedi ISTQB CTFL par. 6.2).

4 Caratteristiche di Qualità del Testing – 60 minuti

Parole Chiave

adattabilità, adeguatezza funzionale, appropriatezza funzionale, capacità di interazione, compatibilità, completezza funzionale, correttezza funzionale, flessibilità installabilità, interoperabilità, testing funzionale, usabilità, user experience

Obiettivi di Apprendimento per il Capitolo 4:

4.1 Testing Funzionale

TA-4.1.1 (K2) Distinguere tra testing di correttezza funzionale, appropriatezza funzionale e completezza funzionale

4.2 Testing di Usabilità

TA-4.2.1 (K2) Spiegare come il Test Analyst contribuisce al testing di usabilità

4.3 Testing di Flessibilità

TA-4.3.1 (K2) Spiegare come il Test Analyst contribuisce al testing di adattabilità e di installabilità

4.4 Testing di Compatibilità

TA-4.4.1 (K2) Spiegare come il Test Analyst contribuisce al testing di interoperabilità

Introduzione alle Caratteristiche di Qualità del Testing

Questo Syllabus utilizza come guida il modello di qualità del prodotto software dello standard ISO 25010 (*ISO/IEC 25010*, 2023) e descrive le caratteristiche di qualità su cui si focalizza un TA. Tuttavia, la terminologia relativa all'usabilità utilizzata differisce da quella di questo standard per essere consistente con il Syllabus ISTQB Certified Tester - Usability Testing (ISTQB CT-UT).

L'Appendice F fornisce una panoramica dell'attuale standard ISO 25010, le modifiche rispetto alla versione precedente e i relativi Syllabi ISTQB che si focalizzano sul testing di specifiche caratteristiche di qualità.

4.1 Testing Funzionale

Il testing funzionale è uno dei compiti principali del TA. Sebbene (ISTQB CTFL) riassume brevemente il testing funzionale come un tipo di test, questo Syllabus entra più nel dettaglio, discutendo le sotto-caratteristiche dell'adeguatezza funzionale.

Lo standard ISO 25010 (*ISO/IEC 25010*, 2023) distingue tre sotto-caratteristiche dell'adeguatezza funzionale. Sebbene tutte possano essere valutate attraverso il testing funzionale, non tutte le attività di test funzionale le indirizzano allo stesso modo. Il TA dovrebbe essere in grado di selezionare i livelli di test e le tecniche di test appropriati per affrontare i rischi di prodotto relativi a una specifica sotto-caratteristica dell'adeguatezza funzionale.

Il testing della completezza funzionale dovrebbe coprire tutte le attività del software specificate e tutti gli obiettivi degli utenti destinatari. La domanda chiave è se tutto ciò che è stato richiesto è stato implementato.

La completezza funzionale dovrebbe essere affrontata il prima possibile, eseguendo la review delle specifiche dei requisiti nei modelli di sviluppo sequenziale e discutendo le user story, compresi i criteri di accettazione, durante la scrittura collaborativa delle user story nello sviluppo software Agile. Nel testing di sistema, nel testing di integrazione dei sistemi e nel testing di accettazione, la completezza funzionale può essere testata dinamicamente.

Le tecniche di test basate sul comportamento, come il testing basato su scenari, sono particolarmente adatte, sebbene siano appropriate anche altre tecniche di test black-box. La tracciabilità tra base di test, condizioni di test e test case è essenziale per determinare il livello di completezza funzionale raggiunto.

Il testing della correttezza funzionale risponde alla domanda se i risultati effettivi sono corretti (p.e. accurati, precisi e consistenti) per input validi e invalidi. È fondamentale trovare un oracolo del test efficace che fornisca i risultati attesi dettagliati.

La correttezza funzionale può essere testata a qualsiasi livello di test. In termini di testing shift-left, la maggior parte del testing di correttezza funzionale dovrebbe essere eseguito durante il testing di componente e il testing di integrazione dei componenti. Anche se il TA non è responsabile di questi livelli di test, dovrebbe comunque contribuire per raggiungere al meglio gli obiettivi del test.

Sono adatte tutte le tecniche di test black-box, le tecniche di test basate sull'esperienza e il testing basato sulla collaborazione.

Il testing di appropriatezza funzionale verifica che le funzioni facilitino il raggiungimento dei compiti e degli obiettivi specificati. Il focus è rivolto al fatto che tutto ciò che è stato implementato soddisfi le esigenze degli utenti.

Il testing di appropriatezza funzionale può includere la review della progettazione della user interface, specialmente per le applicazioni interattive. Il testing dinamico inizia durante il testing di sistema e il testing

di accettazione nei modelli di sviluppo sequenziale, o attraverso sessioni demo nello sviluppo software Agile.

Il testing esplorativo e il testing basato sulla collaborazione sono i più appropriati. Inoltre, sono adatte anche le tecniche di test black-box basate sul comportamento.

4.2 Testing di Usabilità

L'usabilità fa riferimento a un concetto ampio di caratteristiche di qualità legate all'utente, che comprende la capacità di interazione dal modello di qualità del prodotto software (ISO/IEC 25010, 2023) e i benefici dal modello di quality-in-use ISO 25019 (ISO/IEC 25019, 2023). Maggiori informazioni sul testing di usabilità sono disponibili in (ISTQB CT-UT), (ISO 9241-210, 2019) e (UXQB® CPUX-F).

Il testing di usabilità si focalizza generalmente sulla valutazione dei seguenti aspetti:

- capacità di interazione - consentire agli utenti di completare i compiti in contesti d'uso specifici (ISO/IEC 25010, 2023) in modo efficace, efficiente e soddisfacente
- user experience - indirizzare le percezioni degli utenti prima, durante e dopo l'interazione con l'oggetto di test
- accessibilità - garantire che gli utenti con disabilità, background culturali diversi o barriere linguistiche possano utilizzare il sistema in modo efficace ed efficiente

Il TA può collaborare al testing di usabilità già dalle fasi iniziali, mettendo a disposizione la propria conoscenza degli user group target, dei loro obiettivi, del contesto di utilizzo, delle potenziali difficoltà nell'uso del sistema o di user experience negative.

Il TA può contribuire alle principali tecniche di valutazione dell'usabilità come segue:

- **Le review di usabilità** sono eseguite da esperti di usabilità per identificare potenziali problemi di usabilità e deviazioni dai criteri stabiliti. Le review di usabilità possono variare da review informali a ispezioni. Il TA può adattare i criteri di review alle esigenze specifiche degli user group, agli obiettivi e alle priorità di business particolari e al contesto di utilizzo (p.e. tailoring di una checklist di usabilità generica).
- **Le sessioni di testing di usabilità** coinvolgono i futuri utenti o i loro rappresentanti per cercare di risolvere compiti predefiniti e valutare se possono essere completati in modo efficace, efficiente e con soddisfazione. Il TA può contribuire alla progettazione di scenari per le sessioni di testing di usabilità in base a Persona, user group o profili operativi.
- **I questionari o sondaggi utenti**, includono valutazioni e feedback, misurano la soddisfazione degli utenti. Esempi di questionari per gli utenti sono SUMI (Software Usability Measurement Inventory SUMI, 1991) e WAMMI (Website Analysis and Measurement Inventory WAMMI, 1999). Il TA può aiutare a progettare un questionario e a valutare le risposte per indirizzare gli obiettivi specifici degli utenti target nel loro contesto d'uso.

Nel testing di accessibilità, un obiettivo del test comune è verificare la conformità agli standard. Lo standard internazionale Web Content Accessibility Guidelines (WCAG, 2023) definisce tre livelli di conformità (A, AA e AAA) che rappresentano livelli crescenti di accessibilità dei contenuti web. Gli standard nazionali includono Equality Act del Regno Unito (Governo del Regno Unito, 2010) e Americans Disabilities Act degli Stati Uniti (Dipartimento di Giustizia degli Stati Uniti, 2010). Il TA può identificare il livello di conformità richiesto e le esigenze specifiche del gruppo target previsto analizzando il contesto di utilizzo.

4.3 Testing di Flessibilità

Il testing di flessibilità (noto anche come testing di portabilità) verifica che l'oggetto di test possa essere adattato ai cambiamenti nei suoi contesti di utilizzo o nell'ambiente di sistema. Lo standard ISO 25010 (ISO/IEC 25010, 2023) distingue le seguenti sotto-caratteristiche della flessibilità:

- adattabilità
- scalabilità
- installabilità
- sostituibilità

La flessibilità ha aspetti sia tecnici sia non tecnici. Questa sezione si focalizza su come il TA può contribuire al testing di adattabilità e di installabilità. La scalabilità e la sostituibilità sono legate agli aspetti tecnici e sono discusse rispettivamente in (ISTQB CT-PT) e (ISTQB CTAL-TTA).

Il testing di adattabilità verifica che l'oggetto di test possa essere adattato o trasferito all'hardware, al software o ad altri ambienti operativi o di utilizzo previsti.

Il TA supporta il testing di adattabilità identificando gli ambienti target previsti (p.e. le versioni dei sistemi operativi mobile supportati e le versioni dei browser che possono essere utilizzati) e progettando i test che coprano combinazioni di questi ambienti. Poiché questo richiede dati di test rappresentativi di varie configurazioni dei parametri ambientali, vengono spesso applicate tecniche di test come il testing combinatorio (vedi *Sezione 3.1.2*). Un altro esempio è l'integrazione di vari componenti di una piattaforma nei progetti del cliente per garantire la compatibilità tra gli ambienti target. In base al rischio di prodotto, il TA progetta ed esegue dei smoke test o una test suite più completa per verificare che l'oggetto di test sia stato adattato correttamente nell'ambiente target.

Seguendo le buone pratiche del testing di adattabilità (p.e. definendo in anticipo gli ambienti target, utilizzando il testing combinatorio, eseguendo lo smoke testing sui nuovi ambienti e monitorando i difetti specifici dell'ambiente), il TA può rilevare difetti che potrebbero limitare la durata e l'usabilità del software (p.e. la facilità d'uso su schermi di diverse dimensioni). Il lavoro del TA nel testing di adattabilità dovrebbe anche essere supportato da test automatizzati multipiattaforma implementati dai TAE. Un testing di adattabilità rigoroso può rilevare in anticipo problemi specifici dell'ambiente, contribuendo ad evitare frequenti aggiornamenti importanti o riprogettazioni dopo l'implementazione, e riducendo i costi di manutenzione.

Il testing di installabilità verifica che l'oggetto di test possa essere installato, disinstallato, aggiornato e riconfigurato correttamente in ambienti specifici. Il testing di installabilità va oltre il semplice controllo che la procedura di installazione venga eseguita in modo completo con successo.

Gli obiettivi tipici del test di installabilità su cui si focalizza il TA sono:

- Verificare che le procedure di installazione siano eseguite correttamente con diverse configurazioni dei parametri ambientali. Questo rende utili le tecniche di test come il testing combinatorio, in modo simile al testing di adattabilità.
- Progettare ed eseguire test per determinare se l'oggetto di test funziona correttamente dopo l'installazione o l'aggiornamento
- Verificare la facilità con cui gli utenti possono installare, disinstallare o aggiornare il software. Questo include la review della documentazione relativa all'installazione.

- Eseguire il testing del comportamento relativo ai permessi, in particolare per le applicazioni mobile (vedi ISTQB CT-MAT).

4.4 Testing di Compatibilità

Il testing di compatibilità verifica se un oggetto di test è compatibile con altri componenti o sistemi quando viene utilizzato. Lo standard ISO 25010 (*ISO/IEC 25010*, 2023) distingue due sotto-caratteristiche della compatibilità: interoperabilità e coesistenza. Pertanto, il testing di compatibilità può essere suddiviso nei seguenti tipi di test:

- **Testing di interoperabilità**, che verifica la compatibilità con i componenti o i sistemi con cui l'oggetto di test deve interagire. Questi test sono in genere test funzionali black-box, di cui il TA è solitamente responsabile.
- **Testing di coesistenza**, che verifica che l'oggetto di test possa condividere il proprio ambiente target con altri componenti o sistemi senza interferenze. Questo tipo di test tecnico è trattato in (ISTQB CTAL-TTA).

L'obiettivo del testing di interoperabilità è verificare che due o più componenti o sistemi possano scambiarsi informazioni e utilizzare reciprocamente le informazioni scambiate. Quando lo scambio di informazioni comporta una trasformazione dei dati, il testing di interoperabilità deve includere la verifica di tale trasformazione.

Il testing di interoperabilità è particolarmente importante quando più sistemi devono collaborare, condividere dati o eseguire attività congiunte. Questo vale in modo particolare nelle architetture software moderne, come soluzioni cloud, web service, microservizi, containerizzazione e Internet of Things.

L'interoperabilità si verifica generalmente a diversi livelli architetturali. Il TA deve comprendere le possibili interazioni per definire le condizioni di test appropriate a coprirle. Non tutte le interazioni possono essere documentate. Il TA può recuperare indirettamente informazioni sulle interazioni dalla documentazione relativa all'architettura e alla progettazione. È quindi fondamentale comprendere tale documentazione per garantire che tutti gli aspetti importanti delle interazioni vengano testati.

Il testing di interoperabilità può rilevare difetti relativi a:

- trasformazioni dei dati per lo scambio dati
- interpretazione o utilizzo dei dati scambiati
- flussi di comunicazione e protocolli
- conformità agli standard
- funzionalità end-to-end
- documentazione di progettazione

Il testing di interoperabilità viene generalmente applicato durante il testing di integrazione. Le tecniche di test black-box sono adatte al testing di interoperabilità, come le tecniche di test basate sui dati focalizzate sui dati scambiati, le tecniche di test basate sul comportamento per l'interpretazione o l'utilizzo dei dati, e le tecniche rule-based o di funzionalità end-to-end per la trasformazione dei dati. Le tecniche basate sull'esperienza possono essere utili a completare il testing black-box. I test case del testing funzionale o del testing di adattabilità possono essere riutilizzati per il testing di interoperabilità.

Esempi di standard generali in materia di interoperabilità sono (*ISO 15745*, 2003) e (*ISO 16100*, 2009).

ETSI (*ETSI EG 202 237 v1.2.1*, 2010) fornisce un esempio di metodologia concreta di testing di interoperabilità nel dominio delle telecomunicazioni.

5 Prevenzione dei Difetti del Software – 225 minuti

Parole Chiave

lettura perspective-based, prevenzione dei difetti, review ad hoc, review checklist-based, review role-based, review scenario-based, risultato del test, root cause analysis,, tecnica di review, testing model-based

Obiettivi di Apprendimento per il Capitolo 5:

5.1 Pratiche di Prevenzione dei Difetti

TA-5.1.1 (K2) Spiegare in che modo il Test Analyst può contribuire alla prevenzione dei difetti

5.2 Supporto al Contenimento di Fase

TA-5.2.1 (K3) Utilizzare un modello dell'oggetto di test per rilevare i difetti in una specifica

TA-5.2.2 (K3) Applicare una tecnica di review a una base di test per rilevare difetti

5.3 Mitigazione della Ricorrenza dei Difetti

TA-5.3.1 (K4) Analizzare i risultati del test per identificare potenziali miglioramenti nella rilevazione dei difetti

TA-5.3.2 (K2) Spiegare in che modo la classificazione dei difetti supporta la root cause analysis

Introduzione alla Prevenzione dei Difetti del Software

L'obiettivo della prevenzione dei difetti è implementare azioni che riducano la probabilità di (ri)occorrenza dei difetti nei prodotti di lavoro e ne mitigano la propagazione nelle fasi successive dell'SDLC. Questi sforzi producono una serie di importanti benefici, tra cui la riduzione dei costi e dell'effort, l'aumento della produttività e il miglioramento della qualità del prodotto. La prevenzione dei difetti è di responsabilità di tutto il team. Il TA può contribuire utilizzando la propria conoscenza ed esperienza specifica.

Le pratiche di prevenzione dei difetti includono:

- prevenire l'introduzione di difetti, che è parte delle attività di Quality Assurance
- prevenire che i difetti sfuggano alle fasi successive dell'SDLC (vedi par. 5.2)
- prevenire il ripetersi dei difetti (vedi par. 5.3)

5.1 Pratiche di Prevenzione dei Difetti

Il TA può contribuire alla prevenzione dei difetti in diversi modi, utilizzando la propria conoscenza del dominio, la propria esperienza nel testing e le proprie competenze analitiche. Alcuni esempi:

- Partecipare all'analisi del rischio, assicurando che i rischi identificati siano adeguatamente mitigati (p.e. selezionando le tecniche di test più adeguate).
- Eseguire la review dei requisiti, dei modelli e delle specifiche, consentendo la rilevazione anticipata dei difetti nella base di test e prevenendo che questi difetti si propaghino nel codice, con una riduzione significativa dei costi di correzione dei difetti.
- Partecipare alle retrospettive, consentendo l'identificazione di potenziali miglioramenti nell'analisi dei test, nella progettazione dei test, nell'implementazione dei test e nell'esecuzione dei test (p.e. utilizzando tecniche di test più efficaci, indirizzando il testing ad aree specifiche di rischio o migliorando i dati di test e gli ambienti di test per ridurre sia i risultati falsi-positivi sia i risultati falsi-negativi) per prevenire in modo migliore che i difetti sfuggano.
- Raccogliere e valutare i dati dei difetti, supportando la root cause analysis e il process improvement attraverso la raccolta di dati dettagliati sui difetti, in modo da consentirne la classificazione e l'analisi statistica e facilitare così la root cause analysis.
- Partecipare alla root cause analysis, prevenendone la ri-occorrenza attraverso azioni correttive per indirizzare le root cause.

Oltre a partecipare alla prevenzione dei difetti, il TA valuta (di solito con il test manager) se le misure proposte hanno prodotto l'effetto desiderato. Esempi di metriche utili a valutare l'efficacia di queste misure includono:

- Efficienza nella rimozione dei difetti (DRE - Defect Removal Efficiency) – misura il rapporto tra i difetti corretti (eliminati) prima del rilascio e il numero totale di difetti. Il denominatore (non conosciuto) può essere stimato o sostituito dal numero totale di difetti rilevati fino a un determinato momento (p.e. fino a 6 mesi dopo il rilascio). Un DRE elevato indica che pochi difetti si propagano in produzione, il che potrebbe implicare pratiche più efficaci di prevenzione dei difetti. Tuttavia, il DRE non distingue tra difetti intercettati tramite la prevenzione e quelli rilevati.
- Efficacia del contenimento di fase (PCE - Phase Containment Effectiveness) – misura il numero di difetti introdotti e corretti nella stessa fase rispetto al numero totale di difetti introdotti in quella fase.

Un PCE elevato indica che un numero minore di difetti sfugge alle fasi successive.

- Costo della qualità – specifica la relazione tra costi di prevenzione dei difetti, costi di rilevamento dei difetti e costi di correzione dei difetti (vedi ISTQB CTAL-TMpar. 3.2.1).

5.2 Supporto al Contenimento di Fase

L'obiettivo del contenimento di fase è rilevare e correggere i difetti nella stessa fase dell'SDLC in cui sono stati introdotti. Nello sviluppo software Agile, è possibile utilizzare in modo simile l'approccio shift-left. Questa politica riduce il costo della qualità. Poiché la base di test è un input importante per l'analisi dei test e la progettazione dei test, il TA può contribuire al contenimento di fase valutando la qualità della base di test.

L'attenzione anticipata alla qualità della base di test minimizzerà l'effort successivo e impedirà la propagazione dei difetti alle fasi successive. Questo Syllabus discute due opzioni comuni che il TA può utilizzare per trovare difetti nella base di test: utilizzo di modelli per scopi del testing e review della base di test utilizzando varie tecniche di review (ISO/IEC 20246, 2017).

5.2.1 Utilizzo di Modelli per Rilevare Difetti

La modellazione fornisce astrazioni di un sistema con un certo livello di precisione e dettaglio. Aiuta gli stakeholder a comprendere meglio il sistema in fase di sviluppo. Come discusso di seguito, la modellazione può supportare il contenimento di fase in almeno tre modi:

- rilevando difetti nelle specifiche
- rilevando difetti nei modelli
- rilevando difetti negli oggetti di test utilizzando il testing model-based

Rilevare difetti nelle specifiche. Le specifiche sono spesso fornite sotto forma di testo scritto in modo informale. Il TA può rappresentare queste specifiche in modo formale utilizzando modelli. Durante la creazione di un modello, le condizioni di test (p.e. i requisiti) vengono mappate su elementi del modello e collegate ad essi per garantire la tracciabilità. La formalizzazione e la visualizzazione delle condizioni di test in un modello rivelano in modo efficiente difetti quali incompletezza, inconsistenza o ambiguità. Il punto di forza della modellazione è che la specifica viene trasformata mentre nelle review viene verificata nella sua forma originale. Di conseguenza, il TA contribuisce anche alla scoperta di soluzioni appropriate per i difetti rilevati.

I modelli basati sui dati includono modelli di dominio e modelli di testing combinatorio. Consentono al TA di rilevare difetti di dominio come partizioni sovrapposte, lacune nella copertura del dominio, partizioni vuote, combinazioni di parametri mancanti o errate.

I modelli basati sul comportamento includono matrici CRUD, modelli basati sugli stati o modelli di scenario. Consentono al TA di rilevare cicli di vita degli oggetti di test incompleti o inconsistenti, transizioni di stato mancanti o difettose, deadlock, loop infiniti, comportamenti del sistema ambigui o inconsistenti o gestione delle eccezioni mancanti.

I modelli basati su regole, come le tabelle delle decisioni o le relazioni metamorfiche, consentono al TA di rilevare difetti nelle regole di business, come omissioni, inconsistenze, ambiguità, ridondanze, scenari soggetti a errori o logiche di business complesse.

La modellazione può anche rilevare difetti quali inconsistenze nella definizione dei termini, nei valori dei dati (p.e. i limiti) e negli input o output. Può inoltre identificare informazioni mancanti, incomplete, ambigue

o non necessarie.

Rilevare difetti nei modelli. Se la base di test contiene modelli, il TA può analizzare questi modelli per rilevare i difetti. Questo Syllabus discute il rilevamento dei difetti attraverso tre modelli tipici utilizzati nel testing del software: state transition diagram (vedi par. 3.2.2), activity diagram (vedi par. 3.2.3) e tabelle delle decisioni (vedi par. 3.3.1). Esempi di difetti nei modelli sopra menzionati includono:

- state transition diagram: stati mancanti/errati, transizioni improprie, guard condition o azioni non corrette, stati ridondanti o irraggiungibili e comportamento non deterministico
- activity diagram: azioni mancanti, irraggiungibili o senza uscita, ordine errato delle azioni, guard condition errate, non esclusive o incomplete nei nodi decisionali, punti di sincronizzazione mancanti o flussi paralleli sincronizzati in modo improprio che possono portare a comportamenti indesiderati
- tabelle delle decisioni: regole sovrapposte, incoerenti o irrealizzabili, incompletezza (p.e. combinazioni di condizioni mancanti o azioni mancanti per una data combinazione di condizioni)

Tutti i modelli possono contenere anche difetti come errori di sintassi, errori di typing, duplicati e termini inconsistenti degli elementi del modello.

Rilevare i difetti utilizzando il testing model-based (MBT). In questo approccio del test, uno strumento MBT progetta e genera test case e altri testware di progettazione dei test basandosi su un modello MBT e su criteri di selezione dei test definiti dal TA. L'MBT è efficace nella rilevazione di anomalie nella specifica perché consente una copertura completa e un'esplorazione sistematica del comportamento atteso dell'oggetto di test secondo il modello MBT. I modelli MBT, in particolare quelli grafici, favoriscono la comunicazione con gli stakeholder, creando una percezione e una comprensione comuni della base di test. Maggiori dettagli sull'MBT sono disponibili in (ISTQB CT-MBT).

5.2.2 Applicazione delle Tecniche di Review

Una base di test ben definita è fondamentale per garantire la qualità dei prodotti di lavoro e il successo dei progetti. Eseguire la review della base di test aiuta a identificare e indirizzare i difetti in anticipo, prevenendo che si propagino alle fasi successive.

Il TA può impiegare varie tecniche di review durante la review individuale per identificare i difetti nella base di test. La selezione della tecnica di review più appropriata può migliorare l'efficienza e l'efficacia della review. Questa selezione dovrebbe tenere conto di fattori quali gli obiettivi della review, gli obiettivi del progetto, le risorse disponibili, il tipo di base di test, i rischi associati, il dominio di business e la cultura aziendale.

Di seguito, questo Syllabus descrive cinque tecniche di review comunemente utilizzate dal TA.

La review ad hoc viene eseguita dai reviewer in modo informale, senza un processo strutturato. Ai reviewer vengono fornite poche o nessuna indicazione su come dovrebbe essere eseguito il compito. La review ad hoc richiede poca preparazione e dipende in larga misura dalle competenze dei reviewer. Durante la review, i reviewer leggono la base di test e documentano le anomalie man mano che le incontrano. Questa tecnica, se non gestita, può portare a un elevato volume di defect report duplicati da parte di più reviewer.

La review checklist-based prevede la valutazione della base di test rispetto a una checklist predefinita. Le checklist ricordano ai reviewer di verificare punti specifici e possono rendere meno personale la review. Le checklist possono essere generiche o specifiche per caratteristiche di qualità, per obiettivi del test o per tipo di base di test. Il TA adatta la checklist al tipo di base di test, al livello di rischio o alla condizione di test, in modo che la review si focalizzi sugli aspetti più rilevanti della base di test. Le checklist dovrebbero essere aggiornate regolarmente con i difetti precedentemente sfuggiti. Mantenere aggiornate le checklist

impedisce di trascurare anomalie appena identificate. Le checklist non sono esaustive; pertanto, il TA non è limitato alla sola verifica degli elementi elencati. Questo massimizza il rilevamento dei difetti e consente al TA di rilevare anomalie che le checklist possono non coprire esplicitamente.

La **review scenario-based** consiste nel simulare un processo o un'attività per identificare anomalie e raffinare la base di test. Questa tecnica di review è più efficace quando la base di test ha un formato scenario-based, come uno use case o un activity diagram. In questi casi, i reviewer possono eseguire "dry run" basati sull'utilizzo atteso del prodotto di lavoro. Gli scenari forniscono linee guida preziose, ma i reviewer non sono vincolati agli scenari documentati e dovrebbero anche pensare oltre gli scenari per identificare ulteriori anomalie.

La **review role-based** comporta l'assegnazione di specifici ruoli o responsabilità ai reviewer. I ruoli tipici si basano su tipi specifici di utenti finali (p.e. utenti esperti, inesperti, senior o bambini) o su ruoli specifici all'interno dell'organizzazione (p.e. amministratore o utente regolare). Ogni ruolo può essere descritto da una Persona (cioè un personaggio concreto ma fittizio progettato per rappresentare le caratteristiche, le esigenze, gli obiettivi e le preferenze di un particolare gruppo di utenti). Distribuendo le responsabilità tra i reviewer in base ai loro ruoli, le review role-based consentono ai singoli individui di focalizzarsi su aspetti specifici della base di test, garantendo una copertura completa e, allo stesso tempo, evitando la duplicazione delle anomalie.

La **lettura perspective-based** comporta la review della base di test da diverse prospettive o punti di vista (p.e. progettista, tester, commerciale, amministratore e utente finale). Questo porta a una review individuale più approfondita con una minore duplicazione delle anomalie tra i reviewer. Inoltre, la lettura perspective-based richiede ai reviewer di tentare di utilizzare la base di test in fase di review per generare il prodotto di lavoro che ne deriverebbe. Ad esempio, un tester tenterebbe di generare test di accettazione in versione draft basati sulle specifiche dei requisiti, per verificare che tutte le informazioni necessarie siano incluse.

5.3 Mitigazione della Ricorrenza dei Difetti

Un TA può contribuire in modo proattivo a minimizzare la ricorrenza dei difetti nel software. Questo Syllabus discute due approcci relativi alla mitigazione della ricorrenza dei difetti: analizzare i risultati dei test per migliorare l'analisi dei test e la progettazione dei test, supportare la root cause analysis con la classificazione dei difetti.

5.3.1 Analizzare i Risultati del Test per Migliorare il Rilevamento dei Difetti

I risultati del test consentono al TA di identificare i failure, ma forniscono anche un feedback al TA per aiutarlo a migliorare l'efficacia del rilevamento dei difetti. Di seguito sono descritte alcune tecniche comunemente utilizzate per analizzare i risultati dei test.

Analisi dei cluster di difetti previsti rispetto a quelli effettivi. Un piccolo insieme di componenti contiene solitamente la maggior parte dei difetti (vedi ISTQB CTFL par. 1.3). Dopo il testing, il TA può identificare le aree effettivamente soggette a difetti e confrontare i cluster di difetti previsti con quelli effettivi. In caso di discrepanze, è possibile applicare un testing più rigoroso alle aree in cui sono stati rilevati più difetti del previsto. Nel determinare i cluster, criteri misurabili dovrebbero garantire chiarezza e coerenza (p.e. densità dei difetti e severità dei difetti).

Tra questi criteri, la severità dei difetti dovrebbe avere un ruolo significativo. Piccoli cluster di difetti critici o gravi sono solitamente più importanti (cioè richiedono testing più rigoroso) rispetto a cluster più grandi di difetti minori o estetici.

Analisi della percentuale di rilevamento dei difetti (DDP - Defect Detection Percentage). La DDP è

una delle misure più importanti dell'efficacia del testing per un livello di test. Nel calcolare la DDP, il numero di difetti sfuggiti dovrebbe essere limitato a quelli che il livello di test in esame avrebbe potuto rilevare. Per garantire la coerenza, è necessario stabilire confini chiari per il conteggio dei difetti, come limiti temporali (p.e. difetti rilevati entro un determinato periodo di tempo dopo il rilascio) e criteri di esclusione (p.e. difetti in componenti di terze parti o in ambienti specifici dei clienti). Un DDP basso per un dato livello di test indica un'alta percentuale di difetti sfuggiti, il che significa che il rilevamento dei difetti è inefficace. In tal caso, il TA dovrebbe analizzare le ragioni e proporre misure per migliorarlo, rendendo il livello di test più focalizzato e rigoroso. Il DDP è migliore suddiviso per livelli di severità, poiché la priorità di ridurre i difetti sfuggiti dipende generalmente dalla loro severità.

Analisi della copertura strutturale. Valuta in che misura i test hanno esercitato aree specifiche dell'oggetto di test. L'identificazione delle aree a bassa copertura consente al TA di concentrare l'effort del test in queste aree. Aumentarne la copertura aiuta a rilevare nuovi difetti precedentemente sfuggiti. La copertura strutturale, come la copertura delle istruzioni, la copertura dei rami o la copertura dei neuroni, viene generalmente misurata con strumenti di test. Quando si selezionano le aree aggiuntive da coprire, è necessario considerare i loro livelli di rischio.

Analisi delle lacune dei test. Valuta la misura in cui i test hanno esercitato le recenti modifiche al codice. Questo consente al TA di focalizzare ulteriore effort del test alle aree particolarmente soggette a errori (p.e. nuove modifiche che non sono state testate) invece di focalizzarsi a tutte le aree che hanno bassa copertura (p.e. codice che non è stato modificato da molto tempo ed è stato testato per release precedenti).

Analisi dell'andamento dei difetti (defect arrival pattern). Il numero o la densità dei difetti rilevati nelle fasi successive di un progetto (p.e. iterazioni) può essere confrontato con modelli che descrivono la distribuzione teorica di questi valori nel tempo. Un classico esempio di modello di introduzione dei difetti è il modello *di Rayleigh* (Elsayed, 2021). Ha un unico picco ed è inclinato verso destra, mostrando che il numero previsto di difetti rilevati aumenta inizialmente nel tempo e, dopo aver raggiunto il suo valore massimo, scende lentamente verso lo zero. Sulla base dell'analisi di tale modello, è possibile dedurre la capacità dei test case esistenti e il loro potenziale di miglioramento. Ad esempio, si suppone che il rilevamento dei difetti rimanga a un livello basso e costante quando il modello suggerisce che dovrebbe aumentare. In tal caso, questo può significare che i test esistenti sono troppo deboli e non in grado di rilevare ulteriori difetti.

I metodi analitici sopra descritti utilizzano varie metriche relative ai difetti, come il numero di difetti o il DDP. Queste metriche sono calcolate sulla base dei risultati del test (p.e. numero di test superati/falliti), dei defect report e delle metriche strutturali (p.e. copertura del codice o densità dei difetti). Si noti, tuttavia, che queste metriche non sono sempre così facili da leggere dai risultati dei test come potrebbero sembrare. Ad esempio, il numero di test falliti non coincide necessariamente con il numero di difetti rilevati da questi test. Per calcolare il numero effettivo di difetti rilevati, è necessario analizzare attentamente i risultati del processo di debugging, poiché la relazione tra risultati dei test e difetti può essere multi-a-molti. Diversi test possono rilevare lo stesso difetto o un test può rilevare diversi difetti. Inoltre, la severità dei difetti può differire dalla criticità dei test, poiché un test case critico può fallire a causa di un difetto estetico.

5.3.2 Supportare la Root Cause Analysis con la Classificazione dei Difetti

La root cause analysis (RCA) è una tecnica per identificare e indirizzare le cause sottostanti o fondamentali di un difetto piuttosto che solo i suoi sintomi. La RCA supporta un approccio strutturato al miglioramento della qualità e il suo obiettivo principale è prevenire la ricorrenza dei difetti. Il TA utilizza diverse tecniche per identificare le root cause dei difetti e dei failure (p.e. le tassonomie dei difetti, la tecnica dei five whys, i diagrammi causa-effetto e l'analisi di Pareto).

La classica RCA prevede che esperti di dominio studino un difetto in modo molto dettagliato dopo averlo

risolto. Tuttavia, generalmente esistono molti difetti che necessitano di essere analizzati. Pertanto, sarebbe molto inefficiente e dispendioso in termini di tempo pianificare azioni preventive per ogni difetto. Un modo per affrontare il problema è classificare i difetti ed eseguire quindi l'RCA sui tipi di difetti che si manifestano.

La classificazione dei difetti è basata sul concetto che i singoli difetti catturano una grande quantità di informazioni sul processo di sviluppo e sul System Under Test. La classificazione dei difetti consente al TA di estrarre dal difetto le informazioni su vari aspetti del processo di sviluppo e trasformarle in una misurazione di processo. Questa, a sua volta, fornisce una vista dei tipi di errori commessi durante lo sviluppo, utile per il process improvement. La classificazione dei difetti colma il divario tra le statistiche quantitative sui difetti e l'RCA qualitativa. Per supportare efficacemente l'RCA, i difetti dovrebbero essere classificati in modo uniforme durante l'intero SDLC, dal testing anticipato alla produzione.

Il TA dovrebbe supportare la propria organizzazione nella standardizzazione della classificazione dei difetti del software. Questo migliorerà la comunicazione e lo scambio di informazioni sui difetti tra sviluppatori e organizzazioni, facilitando l'RCA.

Esempi di metodi di classificazione dei difetti sono:

- Orthogonal defect classification (ODC) (Chillarege, 1992), che classifica ogni difetto in attributi ortogonali (cioè mutuamente esclusivi), raccolti sia quando un difetto viene segnalato sia quando viene corretto
- IEEE 1044 (*IEEE 1044*, 2009), una classificazione standard per le anomalie del software che fornisce un insieme di attributi fondamentali per la classificazione dei failure e dei difetti
- classificazione basata sulla severità, che classifica i difetti in base alla loro severità (p.e. critici, gravi, minori, triviali)
- modelli di tassonomia dei difetti, come (Beizer, 1990) o (Catolino et al., 2019)

I difetti possono anche essere mappati su attributi di qualità, utilizzando modelli di qualità del software come (*ISO/IEC 25010*, 2023) o il modello FURPS (Grady et al., 1987).

Maggiori informazioni sull'RCA sono disponibili in (ISTQB CTCL-ITP-ITP)

6 Riferimenti

Standard

- ETSI EG 202 237 v1.2.1: Internet Protocol Testing (IPT)*, 2010. Standard. European Telecommunications Standards Institute.
- IEEE 1044: Standard Classification for Software Anomalies*, 2009. Standard. Institute of Electrical and Electronics Engineers.
- ISO 15745: Industrial automation systems and integration – Open systems application integration framework*, 2003. Standard. International Organization for Standardization.
- ISO 16100: Industrial automation systems and integration – Manufacturing software capability profiling for interoperability*, 2009. Standard. International Organization for Standardization.
- ISO 26262: Road vehicles – functional safety – Part 6: Product development at the software level*, 2018. Standard. International Organization for Standardization.
- ISO 9241-210: Ergonomics of human-system interaction, part 210: Human-centred design for interactive systems*, 2019. Standard. International Organization for Standardization.
- ISO/IEC 20246: Software and systems engineering – Work product reviews*, 2017. Standard. International Organization for Standardization.
- ISO/IEC 25010: Systems and software Quality Requirements and Evaluation (SQuaRE) – Product quality model*, 2023. Standard. International Organization for Standardization.
- ISO/IEC 25019: Systems and software Quality Requirements and Evaluation (SQuaRE) – Quality-in-use model*, 2023. Standard. International Organization for Standardization.
- ISO/IEC/IEEE 29119-3: Software testing, Part 3: Test documentation*, 2021. Standard. International Organization for Standardization.
- ISO/IEC/IEEE 29119-4: Software testing, Part 4: Test techniques*, 2021. Standard. International Organization for Standardization.
- ISO/IEC/IEEE 29119-5: Software testing, Part 5: Keyword-Driven Testing*, 2016. Standard. International Organization for Standardization.
- OMG® BPMN: Business Process Model and Notation, version 2.0*, 2013. Standard. Object Management Group, OMG®.
- OMG® DMN: Decision Model and Notation, version 1.5*, 2024. 2024-01. Standard. Object Management Group.
- OMG® UML: Unified Modeling Language, version 2.5*, 2017. Standard. Object Management Group.
- RTCA DO-178C: Software Considerations in Airborne Systems and Equipment Certification*, 2011. Standard. Radio Technical Commission for Aeronautics, RTCA Inc.
- UXQB® CPUX-F: International Usability and User Experience Qualification Board - Certified Professional for Usability and User Experience, Versione 4.01

Documenti ISTQB®

ISTQB CT-ACT, ISTQB®, 2019. *Certified Tester -Acceptance Testing: Syllabus*. Versione 1.0.

ISTQB CT-AI, ISTQB®, 2021. *Certified Tester - AI Testing: Syllabus*. Versione 1.0.

ISTQB CTFL, ISTQB®, 2024. *Certified Tester Foundation Level: Syllabus*. Versione 4.0.1.

ISTQB CTEL-ITP-ITPI, ISTQB®, 2011. *Certified Tester Expert Level Implementing Test Process Improvement: Syllabus*. Versione 1.0.

ISTQB CT-MAT, ISTQB®, 2019. *Certified Tester - Mobile Application Testing: Syllabus*. Versione 1.0.

ISTQB CT-MBT, ISTQB®, 2024. *Certified Tester - Model-Based Tester: Syllabus*. Versione 1.1.

ISTQB CT-PT, ISTQB®, 2018. *Certified Tester - Performance Testing: Syllabus*. Versione 1.0.

ISTQB CTAL-TAE, ISTQB®, 2024. *Certified Tester Advanced Level - Test Automation Engineering: Syllabus*. Versione 2.0.

ISTQB CTAL-TM, ISTQB®, 2024. *Certified Tester Advanced Level - Test Management: Syllabus*. Versione 3.0.

ISTQB CTAL-TTA, ISTQB®, 2021. *Certified Tester Advanced Level - Technical Test Analyst: Syllabus*. Versione 4.0.

ISTQB CT-UT, ISTQB®, 2018. *Certified Tester - Usability Testing: Syllabus*. Versione 1.0.

Libri

AMMANN, Paul; OFFUTT, Jeff, 2008. *Introduction to Software Testing*. Cambridge University Press.

BEIZER, Boris, 1990. *Software Testing Techniques (2nd Ed.)* International Thomson Computer Press.

BINDER, Robert V., 2000. *Testing Object-Oriented Systems*. Addison-Wesley.

ELSAIED, Elsayed A., 2021. *Reliability Engineering*. Wiley.

FORGÁCS, Istvan; KOVÁCS, Attila, 2019. *Practical Test Design: Selection of traditional and automated test design techniques*. BCS, The Chartered Institute for IT.

FORGÁCS, Istvan; KOVÁCS, Attila, 2024. *Modern Software Testing Techniques*. Apress.

GRADY, Robert; CASWELL, Deborah, 1987. *Software Metrics: Establishing a Company-wide Program*. Prentice Hall.

HENDRICKSON, Elisabeth, 2013. *Explore It!* Pragmatic Bookshelf.

JORGENSEN, Paul, 2014. *Software Testing. A Craftsman's Approach*. CRC Press.

KANER, Cem; FALK, Jack; NGUYEN, Hung Q., 1999. *Testing Computer Software*. Wiley.

KANER, Cem; PADMANABHAN, Sowmya; HOFFMAN, Douglas, 2013. *The Domain Testing Workbook*. Context Driven Press.

KOOMEN, Tim; AALST, Leo van der; BROEKMAN, Bart; VROON, Michiel, 2006. *TMap Next for result-driven testing*. UTN Publishers.

KUHN, D. Richard; YU, Lei; KACKER, Raghu N., 2013. *Introduction to Combinatorial Testing*. CRC Press.

Articoli

ALYAHYA, Sultan, 2020. Crowdsourced Software Testing: A Systematic Literature Review. *Information and Software Technology*. Vol. 127, p. 106363. Available from doi: 10.1016/j.infsof.2020.106363.

ANTONIOL, Giuliano; BRIAND, Lionel; DI PENTA, Massimiliano; LABICHE, Yvan, 2002. A case study using the round-trip strategy for state-based class testing. *Proceedings 13th International Symposium on Software Reliability Engineering*, pp. 269–279. ISBN 0-7695-1763-3. Available from doi: 10.1109/ISSRE.2002.1173268.

ARCURI, Andrea; IQBAL, Muhammad Zohaib; BRIAND, Lionel, 2012. Random Testing: Theoretical Results and Practical Implications. *IEEE Transactions on Software Engineering*. Vol. 38, pp. 258–277. Available from doi: 10.1109/TSE.2011.121.

BARR, Earl; HARMAN, Mark; MCMINN, Phil; SHAHBAZ, Muzammil; YOO, Shin, 2014. The Oracle Problem in Software Testing: A Survey. *IEEE Transactions on Software Engineering*. Vol. 41, no. 5, pp. 507–525. Available from doi: 10.1109/TSE.2014.2372785.

BOCHMANN, Gregor; PETRENKO, Alexandre, 1994. Protocol Testing: Review of Methods and Relevance for Software Testing. *ISSTA '94: Proceedings of the 1994 ACM SIGSOFT international symposium on Software testing and analysis*, pp. 109–124. Available from doi: 10.1145/186258.187153.

CATOLINO, Gemma; PALOMBA, Fabio; ZAIDMAN, Andy; FERRUCCI, Filomena, 2019. Not All Bugs Are the Same: Understanding, Characterizing, and Classifying Bug Types. *Journal of Systems and Software*. Vol. 152, pp. 165–181. Available from doi: 10.1016/j.jss.2019.03.002.

CHILLAREGE, R. et al., 1992. Orthogonal Defect Classification - A Concept for In-Process Measurements. *IEEE Transactions on Software Engineering*. Vol. 18, pp. 943–956. Available from doi: 10.1109/32.177364.

CHOW, Tsun, 1978. Testing Software Design Modeled by Finite-State Machines. *IEEE Transactions on Software Engineering*. Vol. 4, pp. 178–187. Available from doi: 10.1109/TSE.1978.231496.

COHEN, D.M.; DALAL, Siddhartha; KAJLA, A.; PATTON, G.C., 1994. The Automatic Efficient Test Generator (AETG) system. *Proceedings of 1994 IEEE International Symposium on Software Reliability Engineering*, pp. 303–309. ISBN 0-8186-6665-X. Available from doi: 10.1109/ISSRE.1994.341392.

ENGSTRÖM, Emelie; RUNESON, Per; SKOGLUND, Mats, 2010. A systematic review on regression test selection techniques. *Information and Software Technology*. Vol. 52, pp. 14–30. Available from doi: 10.1016/j.infsof.2009.07.001.

GHAZI, Ahmad Nauman; GARIGAPATI, Ratna; PETERSEN, Kai, 2017. Checklists to Support Test Charter Design in Exploratory Testing. *Agile Processes in Software Engineering and Extreme Programming. XP 2017*. ISBN 978-3-319-57632-9. Available from doi: 10.1007/978-3-319-57633-6_17.

HAREL, David, 1987. Statecharts: A Visual Formalism For Complex Systems. *Science of Computer Programming*. Vol. 8, pp. 231–274. Available from doi: 10.1016/0167-6423(87)90035-9.

HUANG, Rubing; SUN, Weifeng; XU, Yinyin; CHEN, Haibo; TOWEY, Dave; XIA, Xin, 2019. A Survey on Adaptive Random Testing. *IEEE Transactions on Software Engineering*. Vol. 47, no. 10, pp. 2052–2083. Available from doi: 10.1109/TSE.2019.2942921.

- JENG, Bingchiang; WEYUKER, Elaine, 1994. A Simplified Domain-Testing Strategy. *ACM Transactions on Software Engineering and Methodology*. Vol. 3, pp. 254–270. Available from doi: 10.1145/196092.193171.
- JUERGENS, Elmar; PAGANO, Dennis; GOEB, Andreas, 2018. Test Impact Analysis: Detecting Errors Early Despite Large, Long-Running Test Suites. *Whitepaper, CQSE GmbH*.
- KUHN, D.; WALLACE, Dolores; JR, A.M., 2004. Software Fault Interactions and Implications for Software Testing. *IEEE Transactions on Software Engineering*. Vol. 30, pp. 418–421. Available from doi: 10.1109/TSE.2004.24.
- LEICHT, Niklas; BLOHM, Ivo; LEIMEISTER, Jan Marco, 2017. Leveraging the Power of the Crowd for Software Testing. *IEEE Software*. Vol. 34, pp. 62–69. Available from doi: 10.1109/MS.2017.37.
- OFFUTT, A. Jefferson, 1992. Investigations of the software testing coupling effect. *ACM Transactions on Software Engineering and Methodology*. Vol. 1, pp. 5–20.
- RECHTBERGER, Vaclav; BURES, Miroslav; AHMED, Bestoun, 2022. Overview of Test Coverage Criteria for Test Case Generation from Finite State Machines Modelled as Directed Graphs. *IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), Valencia, Spain*, pp. 207–214.
- RWEMALIKA, Renaud; KINTIS, Marinos; PAPADAKIS, Mike; LE TRAON, Yves; LORRACH, Pierre, 2019. On the Evolution of Keyword-Driven Test Suites. *12th IEEE Conference on Software Testing, Validation and Verification (ICST)*, pp. 335–345.
- SEGURA, Sergio; FRASER, Gordon; SANCHEZ, Ana; RUIZ-CORTÉS, Antonio, 2016. A Survey on Metamorphic Testing. *IEEE Transactions on Software Engineering*, pp. 46–53.
- SEGURA, Sergio; TOWEY, Dave; ZHOU, Zhi Quan; CHEN, Tsong Yueh, 2020. Metamorphic Testing: Testing the Untestable. *IEEE Software*. Vol. 42, no. 9, pp. 805–824.
- WU, Huayao; NIE, Changhai; PETKE, Justyna; JIA, Yue; HARMAN, Mark, 2020. An Empirical Comparison of Combinatorial Testing, Random Testing and Adaptive Random Testing. *IEEE Transactions on Software Engineering*. Vol. 46, no. 3, pp. 302–320.

Pagine web

- COMMISSION, European, 2016. *General Data Protection Regulation: Regulation (EU) 2016/679* [online]. [visited on 2024-09-15]. Available from: https://commission.europa.eu/law/law-topic/data-protection/legal-framework-eu-data-protection_en?
- COMPUTER SOCIETY, IEEE; JTC 1/SC7, ISO/IEC, 2024. *SE VOCAB: Software and Systems Engineering Vocabulary* [online]. [visited on 2024-09-15]. Available from: <https://pascal.computer.org/>.
- CZERWONKA, Jacek, 2004. *Pairwise Testing: Combinatorial Test Case Generation* [online]. [visited on 2024-09-15]. Available from: www.pairwise.org.
- HEALTH, U.S. Department of; SERVICES, Human, 2024. *Health Insurance Portability and Accountability Act: Standards for Privacy of Individually Identifiable Health Information (Privacy Rules)* [online]. [visited on 2024-09-15]. Available from: <https://www.hhs.gov/hipaa/for-professionals/privacy/laws-regulations/index.html>.
- IREB-GLOSSARY, IREB®, 2024. *The CPRE Glossary: Core terms of Requirements Engineering* [online]. [visited on 2024-09-15]. Available from: <https://glossary.istqb.org>.

ISTQB-GLOSSARY, ISTQB®, 2024. *Standard Glossary of Terms Used in Software Testing* [online]. [visited on 2024-09-15]. Available from: <https://glossary.istqb.org>.

Site of Software Test Design, 2020 [online]. [visited on 2024-09-15]. Available from: <https://test-design.org/>.

SUMI, 1991. *Software Usability Measurement Inventory: Standard evaluation questionnaire for assessing quality of use* [online]. [visited on 2024-09-15]. Available from: sumi.uxp.ie.

U.S. DEPARTMENT OF JUSTICE, Civil Rights Division, 2010. *Americans with Disabilities Act: Guidance & Resource Materials* [online]. [visited on 2024-09-15]. Available from: www.ada.gov/resources/.

UK GOVERNMENT, Equalities Office, 2010. *Equality Act 2010: guidance: Information and guidance on the Equality Act 2010, including age discrimination and public sector Equality Duty*. [online]. [visited on 2024-09-15]. Available from: www.gov.uk/guidance/equality-act-2010-guidance.

WAMMI, 1999. *Website Analysis and Measurement Inventory: Professional service for analyzing user experience* [online]. [visited on 2024-09-15]. Available from: www.wammi.com.

WCAG, 2023. *Web Content Accessibility Guidelines 2.2: W3C Recommendation* [online]. [visited on 2024-09-15]. Available from: www.w3.org/TR/WCAG22/.

I riferimenti precedenti rimandano a informazioni disponibili su Internet e altrove. Sebbene tali riferimenti siano stati verificati al momento della pubblicazione del presente Syllabus, ISTQB® non può essere ritenuto responsabile qualora i riferimenti non fossero più disponibili.

7 Appendice A – Obiettivi di Apprendimento/Livello Cognitivo di Conoscenza

Gli obiettivi di apprendimento (LO) specifici che si applicano a questo Syllabus sono indicati all'inizio di ogni capitolo. Ogni argomento del Syllabus sarà esaminato in base al suo obiettivo di apprendimento.

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

Livello 1: Ricordare (K1)

Il candidato è in grado di ricordare, riconoscere e richiamare un termine o un concetto.

Verbi di azione: richiamare, riconoscere.

Nota: i Syllabi Advanced Level non hanno specifici obiettivi di apprendimento di livello K1. Tuttavia, il contenuto del Syllabus nei capitoli da 1 a 5, e le definizioni di tutti i termini elencati come parole chiave sotto i titoli dei capitoli devono essere ricordati (K1), anche se non esplicitamente menzionati negli obiettivi di apprendimento.

Livello 2: Comprendere (K2)

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

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

Esempi	Note
Distinguere tra testing di correttezza funzionale, appropriatezza funzionale e completezza funzionale	Cercare le differenze tra i concetti.
Spiegare il testing CRUD	
Fornire esempi di requisiti dell'ambiente di test	
Riassumere il coinvolgimento del Test Analyst nei vari cicli di vita dello sviluppo software	

Livello 3: Applicare (K3)

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

Verbi di azione: applicare, implementare, preparare, utilizzare

Esempi	Note
Applicare il testing di dominio	Dovrebbe riferirsi a una procedura / tecnica / processo ecc.
Preparare test charter per il testing session-based	
Utilizzare il testing keyword-driven per sviluppare test script	Può essere utilizzato in un obiettivo di apprendimento che richiede al candidato di essere in grado di utilizzare una tecnica o una procedura.

	Simile a "applicare".
--	-----------------------

Livello 4: Analizzare (K4)

Il candidato è in grado di separare le informazioni relative a una procedura o a una tecnica nelle sue parti costituenti per una migliore comprensione, ed è in grado di distinguere tra fatti e inferenze. Un'applicazione tipica è analizzare un documento, un software o una situazione di progetto, e proporre azioni appropriate per risolvere un problema o completare un compito.

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

Esempi	Note
Analizzare l'impatto delle modifiche per determinare l'ambito del regression testing	Esaminabile solo in combinazione con un obiettivo misurabile dell'analisi. Dovrebbe essere della forma "Analizzare X per Y" (o simile).
Selezionare tecniche di test appropriate per mitigare i rischi di prodotto per una determinata situazione	Necessario quando la selezione richiede un'analisi.

Riferimento

(Per i livelli cognitivi degli obiettivi di apprendimento)

Anderson, L. W. e Krathwohl, D. R. (a cura di) (2001) A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives, Allyn & Bacon

8 Appendice B – Matrice di Tracciabilità dei Business Outcome rispetto agli Obiettivi di Apprendimento

Questa sezione elenca la tracciabilità tra i Business Outcome e gli Obiettivi di Apprendimento del Syllabus Certified Tester Advanced Level - Test Analyst.

Business Outcome: Advanced Level Test Analyst		TA-BO1	TA-BO2	TA-BO3	TA-BO4	TA-BO5	TA-BO6	TA-BO7	TA-BO8	TA-BO9
TA-BO1	Supportare ed eseguire test appropriati basati sul ciclo di vita dello sviluppo software seguito	6								
TA-BO2	Applicare i principi del testing basato sul rischio		3							
TA-BO3	Selezionare e applicare tecniche di test appropriate per supportare il raggiungimento degli obiettivi del test.			13						
TA-BO4	Fornire documentazione con livelli adeguati di dettaglio e qualità				5					
TA-BO5	Determinare i tipi appropriati di testing funzionale da eseguire					1				
TA-BO6	Contribuire al testing non-funzionale						3			
TA-BO7	Contribuire alla prevenzione dei difetti							5		
TA-BO8	Migliorare l'efficienza del processo di test con l'utilizzo di strumenti								4	
TA-BO9	Specificare i requisiti per gli ambienti di test e i dati di test.									2

Business Outcome: Advanced Level Test Analyst		TA-BO1	TA-BO2	TA-BO3	TA-BO4	TA-BO5	TA-BO6	TA-BO7	TA-BO8	TA-BO9
LO-ID	Obiettivo di Apprendimento (livello K)									
1	I compiti del Test Analyst nel processo di test – 225 minuti									
1.1	Testing nel ciclo di vita dello sviluppo software									
TA-1.1.1	Riassumere il coinvolgimento del Test Analyst nei vari cicli di vita dello sviluppo software (K2)	×								
1.2	Coinvolgimento nelle attività di test									
TA-1.2.1	Riassumere i compiti svolti dal Test Analyst nell'ambito dell'analisi dei test (K2)	×								
TA-1.2.2	Riassumere le attività svolte dal Test Analyst nell'ambito della progettazione dei test (K2)	×								
TA-1.2.3	Riassumere le attività svolte dal Test Analyst nell'ambito dell'implementazione dei test (K2)	×								
TA-1.2.4	Riassumere le attività svolte dal Test Analyst nell'ambito dell'esecuzione dei test (K2)	×								
1.3	Compiti relativi ai prodotti di lavoro									
TA-1.3.1	Distinguere tra test case di alto livello e test case di basso livello (K2)				×					
TA-1.3.2	Spiegare i criteri di qualità per i test case (K2)				×					
TA-1.3.3	Fornire esempi di requisiti dell'ambiente di test (K2)				×					×
TA-1.3.4	Spiegare il problema dell'oracolo del test e le potenziali soluzioni (K2)			×	×					
TA-1.3.5	Fornire esempi di requisiti dei dati di test (K2)				×					×
TA-1.3.6	Utilizzare il testing keyword-driven per sviluppare test script (K3)	×							×	
TA-1.3.7	Riassumere i tipi di strumenti per gestire il testware (K2)								×	
2	I compiti del Test Analyst nel testing basato sul rischio – 90 minuti									

Business Outcome: Advanced Level Test Analyst		TA-BO1	TA-BO2	TA-BO3	TA-BO4	TA-BO5	TA-BO6	TA-BO7	TA-BO8	TA-BO9
2.1	Analisi del rischio									
TA-2.1.1	Riassumere il contributo del Test Analyst all'analisi del rischio di prodotto (K2)		X							
2.2	Controllo del rischio									
TA-2.2.1	Analizzare l'impatto delle modifiche per determinare l'ambito del regression testing (K4)		X						X	
3	Analisi dei test e progettazione dei test – 615 minuti									
3.1	Tecniche di test basate sui dati									
TA-3.1.1	Applicare il testing di dominio (K3)			X						
TA-3.1.2	Applicare il testing combinatorio (K3)			X						
TA-3.1.3	Riassumere i vantaggi e i limiti del testing casuale (K2)			X						
3.2	Tecniche di test basate sul comportamento									
TA-3.2.1	Spiegare il testing CRUD (K2)			X						
TA-3.2.2	Applicare il testing delle transizioni di stato (K3)			X						
TA-3.2.3	Applicare il testing basato su scenari (K3)			X						
3.3	Tecniche di test basate su regole									
TA-3.3.1	Applicare il testing della tabella delle decisioni (K3)			X						
TA-3.3.2	Applicare il testing metamorfico (K3)			X						
3.4	Testing basato sull'esperienza									
TA-3.4.1	Preparare i test charter per il testing session-based (K3)			X						
TA-3.4.2	Preparare le checklist a supporto del testing basato sull'esperienza (K3)			X						

Business Outcome: Advanced Level Test Analyst		TA-BO1	TA-BO2	TA-BO3	TA-BO4	TA-BO5	TA-BO6	TA-BO7	TA-BO8	TA-BO9
TA-3.4.3	Fornire esempi dei vantaggi e dei limiti del crowd testing (K2)			X						
3.5	Applicazione delle tecniche di test più appropriate									
TA-3.5.1	Selezionare tecniche di test appropriate per mitigare i rischi di prodotto in una determinata situazione (K4)		X	X						
TA-3.5.2	Spiegare i vantaggi e i rischi dell'automazione della progettazione dei test (K2)								X	
4	Caratteristiche di qualità del testing – 60 minuti									
4.1	Testing funzionale									
TA-4.1.1	Distinguere tra testing di correttezza funzionale, appropriatezza funzionale e completezza funzionale (K2)					X				
4.2	Testing di usabilità									
TA-4.2.1	Spiegare come il Test Analyst contribuisce al testing di usabilità (K2)						X			
4.3	Testing di flessibilità									
TA-4.3.1	Spiegare come il Test Analyst contribuisce al testing di adattabilità e di installabilità (K2)						X			
4.4	Testing di compatibilità									
TA-4.4.1	Spiegare come il Test Analyst contribuisce al testing dell'interoperabilità (K2)						X			
5	Prevenzione dei difetti del software – 225 minuti									
5.1	Pratiche di prevenzione dei difetti									
TA-5.1.1	Spiegare come il Test Analyst può contribuire alla prevenzione dei difetti (K2)							X		
5.2	Supporto al contenimento di fase									
TA-5.2.1	Utilizzare un modello dell'oggetto di test per rilevare difetti in una specifica (K3)							X		

Business Outcome: Advanced Level Test Analyst		TA-BO1	TA-BO2	TA-BO3	TA-BO4	TA-BO5	TA-BO6	TA-BO7	TA-BO8	TA-BO9
TA-5.2.2	Applicare una tecnica di review a una base di test per rilevare difetti (K3)							X		
5.3	Mitigazione della ricorrenza dei difetti									
TA-5.3.1	Analizzare i risultati del test per identificare potenziali miglioramenti nella rilevazione dei difetti (K4)							X		
TA-5.3.2	Spiegare in che modo la classificazione dei difetti supporta la root cause analysis (K2)							X		

9 Appendice C – Release Note

Il Syllabus ISTQB® Certified Tester Advanced Level - Test Analyst (CTAL-TA) v4.0 è un major release. Per questo motivo, non sono disponibili release note dettagliate per ogni capitolo e paragrafo. Tuttavia, di seguito è riportato un riepilogo delle principali modifiche. Inoltre, in un documento separato denominato Release Notes, ISTQB® fornisce la tracciabilità dettagliata tra gli obiettivi di apprendimento delle versioni v3.1.2 e v4.0 del Syllabus CTAL-TA.

Questa major release ha apportato le seguenti modifiche:

Struttura del Syllabus

In questa versione tutti gli Obiettivi di Apprendimento (LO) sono stati modificati per renderli atomici, e per creare una tracciabilità uno-a-uno tra Obiettivi di Apprendimento e contenuti, in modo da evitare di avere un contenuto senza un corrispondente Obiettivo di Apprendimento. Ogni obiettivo di apprendimento è descritto in un paragrafo con lo stesso numero (p.e. TA-1.1.1 è discusso nel par. 1.1.1). L'obiettivo è rendere questa versione più facile da leggere, comprendere, imparare e tradurre, focalizzandosi sull'aumento dell'utilità pratica e sull'equilibrio tra conoscenze e competenze.

Esistono 36 obiettivi di apprendimento nella versione v4.0 rispetto ai 31 obiettivi della versione v3.1.2. Diversi obiettivi di apprendimento K4 sono stati ridotti a K2 o K3. Nel caso degli obiettivi di apprendimento relativi alle tecniche di test, la motivazione è di focalizzare l'attenzione sull'applicazione delle tecniche di test e non sull'analisi della documentazione per l'ulteriore progettazione dei test. Alcuni obiettivi di apprendimento sono stati uniti in un unico obiettivo. La tabella seguente mostra statistiche dettagliate sul numero di obiettivi di apprendimento e sul tempo di formazione.

Versione del Syllabus	#K2 LO (tempi)	#K3 LO (tempi)	#K4 LO (tempi)	#Totale LO (tempi)
attuale (4.0)	22 (330 min)	11 (660 min)	3 (225 min)	36 (1215 min)
precedente (3.1.2)	16 (240 min)	5 (300 min)	10 (750 min)	31 (1290 min)

Classificazione delle tecniche di test

La struttura del Capitolo 3 riflette una classificazione più dettagliata delle tecniche di test rispetto alla versione v3.1.2. Le tecniche di test black-box sono ora categorizzate come tecniche di test basate sui dati, basate sul comportamento e basate sulle regole, a seconda del tipo di modello dell'oggetto di test sottostante.

Estensione dell'ambito del capitolo 5

Il vecchio capitolo 5 (dedicato esclusivamente alle review) è stato ampliato per discutere altre forme fondamentali di pratiche di prevenzione dei difetti utilizzate dal TA, come l'uso di modelli per rilevare difetti nelle specifiche, l'analisi dei risultati del test per migliorare il rilevamento dei difetti e l'uso della classificazione dei difetti per supportare la root cause analysis.

Modifiche agli obiettivi di apprendimento

- Il capitolo 1 è stato riorganizzato in un paragrafo dedicato alle attività di test e un paragrafo dedicato al testware.
- L'argomento relativo ai test case di alto livello e ai test case di basso livello (vecchio TA-1.4.2) è stato declassato da K4 a K2 (TA-1.3.1).

- L'obiettivo di apprendimento K3 sul testing basato sul rischio (vecchio TA-2.1.1) è stato diviso in due, K2 TA-2.1.1 relativo all'analisi del rischio e K4 TA-2.2.1 relativo al controllo del rischio.
- Il partizionamento di equivalenza (vecchio K4 TA-3.2.1) e l'analisi ai valori limite (vecchio K4 TA-3.2.2) sono stati sostituiti dal più generico K3 TA-3.1.1 sul testing di dominio.
- Il testing pairwise (vecchio K4 TA-3.2.6) e i diagrammi ad albero di classificazione (vecchio K2 TA-3.2.5) sono stati uniti in un unico elemento più generale, K3 TA-3.1.2 sul testing combinatorio.
- Il testing delle transizioni di stato (vecchio K4 TA-3.2.4) è stato sostituito con K3 TA-3.2.2, che si focalizza sulla copertura N-switch e round-trip. Sono state eliminate le ridondanze con CTFL.
- Il testing degli use case (vecchio K4 TA-3.2.7) è stato sostituito con il più generale K3 TA-3.2.3 sul testing basato su scenari.
- Il testing della tabella delle decisioni (vecchio K4 TA-3.2.3) è stato declassato a K3 (TA-3.3.1).
- Il testing esplorativo (vecchio K3 TA-3.3.2) è stato sostituito con due obiettivi di apprendimento separati: uno relativo alla preparazione dei test charter (K3 TA-3.4.1) e uno relativo alla preparazione delle checklist a supporto del testing basato sull'esperienza (K3 TA-3.4.2). Le ridondanze con l'attuale CTFL sono state eliminate.
- Quattro obiettivi di apprendimento relativi al confronto delle tecniche di test e all'applicazione della tecnica più appropriata (precedentemente: K4 TA-3.2.8, K2 TA-3.3.3, K2 TA-3.4.1, K2 TA-3.3.1) sono stati combinati in un unico K4 TA-3.5.1.
- Quattro obiettivi di apprendimento relativi al testing funzionale (precedentemente K2 TA-4.2.1, K2 TA-4.2.2, K2 TA-4.2.3 e K4 TA-4.2.7) sono stati combinati in un unico K2 TA-4.1.1. L'argomento è stato semplificato poiché il testing funzionale è già ampiamente descritto nel contesto delle tecniche di test nel Capitolo 3.
- Gli argomenti del Capitolo 6 (Strumenti di test) sono stati spostati nel paragrafo 1.3, focalizzandosi su argomenti più pratici. Il vecchio K3 TA-6.2.1 "Per un dato scenario, determinare le attività appropriate per un Test Analyst in un progetto di testing keyword-driven" è stato leggermente modificato in K3 TA-1.3.6 "Utilizzare il testing keyword-driven per sviluppare test script". Il vecchio K2 TA-6.3.1 "Spiegare l'uso e i tipi di strumenti di test applicati nella progettazione dei test, nella preparazione dei dati di test e nell'esecuzione dei test" è stato leggermente modificato in K2 TA-1.3.7 "Riassumere i tipi di strumenti applicati nella gestione del testware".
- Due obiettivi di apprendimento simili sulle review (vecchio K3 TA-5.2.1 e K3 TA-5.2.2) sono stati combinati in un unico K3 TA-5.2.2.

Nuovi argomenti

- Criteri di qualità per i test case (K2 TA-1.3.2)
- Requisiti dell'ambiente di test (K2 TA-1.3.3)
- Identificazione degli oracoli del test (K2 TA-1.3.4)
- Requisiti dei dati di test (K2 TA-1.3.5)
- Testing casuale (K2 TA-3.1.3)
- Testing CRUD (K2 TA-3.2.1)
- Testing metamorfico (K3 TA-3.3.2)

- Crowd testing (K2 TA-3.4.3)
- Vantaggi e rischi dell'automazione della progettazione dei test (K2 TA-3.5.2)
- Contributi del Test Analyst alla prevenzione dei difetti (K2 TA-5.1.1)
- Utilizzo di modelli per rilevare difetti nelle specifiche (K3 TA-5.2.1)
- Analisi dei risultati del test per migliorare il rilevamento dei difetti (K4 TA-5.3.1)
- Supporto alla root cause analysis attraverso la classificazione dei difetti (K2 TA-5.3.2)

Aggiornamenti degli standard

Le modifiche apportate alle ultime versioni degli standard internazionali per la qualità del software (*ISO/IEC 25010*, 2023) e per le tecniche di test (*ISO/IEC/IEEE 29119-4*, 2021) hanno portato all'adeguamento dei contenuti corrispondenti del Syllabus.

10 Appendice D – Elenco delle abbreviazioni

Abbreviazione	Significato dell'acronimo	Traduzione dell'acronimo
AI	artificial intelligence	intelligenza artificiale
BPMN	Business Process Model and Notation	Business Process Model and Notation
BVA	boundary value analysis	analisi ai valori limite
CRUD	create, read, update, and delete	crea, leggi, aggiorna ed elimina
CSV	comma-separated value	valori separati da virgola
DDP	defect detection percentage	defect detection percentage
DRE	defect removal efficiency	defect removal efficiency
EP	equivalence partitioning	partizionamento di equivalenza
GDPR	General Data Protection Regulation	General Data Protection Regulation
JSON	JavaScript Object Notation	JavaScript Object Notation
MBT	model-based testing	testing model-based
MR	metamorphic relation	relazione metamorfica
MT	metamorphic testing	testing metamorfico
ODC	orthogonal defect classification	classificazione ortogonale dei difetti
PCE	phase containment effectiveness	efficacia del contenimento di fase
RCA	root cause analysis	analisi delle root cause
SDLC	software development lifecycle	ciclo di vita dello sviluppo software
TA	test analyst	test analyst
TTA	technical test analyst	technical test analyst
UML	Unified Modeling Language	Unified Modeling Language
UX	user experience	user experience
XML	Extensible Markup Language	Extensible Markup Language

11 Appendice E – Termini specifici del dominio

Termine	Definizione
Matrice CRUD	Una matrice che indica i tipi di azione delle funzioni sulle entità all'interno di un sistema.
semantica dei dati	Il significato e l'interpretazione dei dati.
tecnica five whys	Una tecnica interrogativa iterativa utilizzata per determinare la root cause di un difetto o di un problema ripetendo cinque volte la domanda "perché?", ogni volta indirizzando il "perché" corrente alla risposta del "perché" precedente.
Analisi di Pareto	Un approccio per identificare le aree problematiche o le attività che avranno maggiore probabilità di accadimento.
user journey map	Documentazione di visualizzazione della user experience che mostra i passi che un utente compie in un processo per raggiungere un obiettivo.
user research	Una disciplina che studia le esigenze e i processi mentali degli utenti osservando il modo in cui svolgono le attività, interagiscono con un componente o un sistema, oppure analizzando e interpretando i dati.

12 Appendice F – Modello di Qualità del Software

La seguente tabella visualizza le caratteristiche del modello di qualità del prodotto ISO/IEC 25010 (ISO/IEC 25010, 2023). Questo standard indica quali caratteristiche/sotto-caratteristiche sono indirizzate in questo Syllabus (CTAL-TA) e quali sono trattate in altri Syllabi ISTQB® (CTAL-TTA, CT-PT, CT-UT, CT- AuT e CT-SEC). Se più Syllabi trattano una caratteristica di qualità, viene elencato per primo quello che la tratta in modo più dettagliato. La tabella confronta anche l'attuale modello ISO/IEC 25010 con la versione 2011 utilizzata nella versione precedente di questo Syllabus.

ISO/IEC 25010:2023 (attuale)	ISO/IEC 25010:2011 (precedente)	Note	Syllabus ISTQB®
Adeguatezza funzionale	Adeguatezza funzionale		CTAL-TA
Completezza funzionale	Completezza funzionale		
Correttezza funzionale	Correttezza funzionale		
Appropriatezza funzionale	Appropriatezza funzionale		
Efficienza delle prestazioni	Efficienza delle prestazioni		CT-PT, CTAL-TTA
Comportamento nel tempo	Comportamento nel tempo		
Utilizzo delle risorse	Utilizzo delle risorse		
Capacità	Capacità		
Compatibilità	Compatibilità		CTAL-TA, CTAL-TTA
Coesistenza	Coesistenza		CTAL-TTA
Interoperabilità	Interoperabilità		CTAL-TA
Capacità di interazione	Usabilità	Rinominato	CT-UT, CTAL-TA
Riconoscibilità dell'appropriatezza	Riconoscibilità dell'appropriatezza		
Apprendibilità	Apprendibilità		
Operabilità	Operabilità		
Protezione da errori dell'utente	Protezione da errori dell'utente		
Coinvolgimento dell'utente	Estetica della user interface	Rinominato	
Inclusività	Accessibilità	Diviso in due parti e rinominato	
Assistenza utente			
Autodescrizione		Nuovo	

ISO/IEC 25010:2023 (attuale)	ISO/IEC 25010:2011 (precedente)	Note	Syllabus ISTQB®
Affidabilità	Affidabilità		CTAL-TTA
Assenza di difetti	Maturità	Rinominato	
Disponibilità	Disponibilità		
Tolleranza ai guasti	Tolleranza ai guasti		
Recuperabilità	Recuperabilità		
Sicurezza	Sicurezza		CT-SEC, CTAL-TTA
Confidenzialità	Confidenzialità		
Integrità	Integrità		
Non-repudiation	Non-repudiation		
Responsabilità	Responsabilità		
Autenticità	Autenticità		
Resistenza		Nuovo	
Manutenibilità	Manutenibilità		CTAL-TTA
Modularità	Modularità		
Riutilizzabilità	Riutilizzabilità		
Analizzabilità	Analizzabilità		
Modificabilità	Modificabilità		
Testabilità	Testabilità		
Flessibilità	Portabilità	Rinominato	CTAL-TTA, CTAL-TA, CT-PT
Adattabilità	Adattabilità		CTAL-TA, CTAL-TTA
Scalabilità		Nuovo	CT-PT
Installabilità	Installabilità		CTAL-TA, CTAL-TTA
Sostituibilità	Sostituibilità		CTAL-TTA
Sicurezza		Nuovo	CT-AuT
Vincolo operativo		Nuovo	
Identificazione del rischio		Nuovo	
Sicurezza intrinseca		Nuovo	
Avviso di pericolo		Nuovo	
Integrazione sicura		Nuovo	

13 Appendice G – Marchi commerciali

ISTQB® è un marchio registrato di International Software Testing Qualifications Board.

UML® è un marchio registrato di Object Management Group (OMG).

BPMN™ è un marchio commerciale di Object Management Group (OMG).

14 Indice analitico

accessibilità (46)
activity diagram (35, 53, 54)
adattabilità (14, 39, 44, 47, 48)
analisi ai valori limite (30, 42)
analisi degli impatti (15, 24, 26, 27)
analisi del rischio (16, 24, 25, 42, 51)
analisi dell'andamento dei difetti (55)
analisi di Pareto (55, 73)
appropriatezza funzionale (44, 45, 62)
chaos engineering (33)
checklist (25, 28, 39, 40, 46)
checklist do-confirm (40)
checklist read-do (40)
cluster di difetti (17, 54)
coesistenza (48)
configuration item (26)
confine (30, 31)
confine aperto (30, 31)
confine chiuso (30, 31)
contenimento di fase (50, 52)
copertura (15, 17, 19, 25, 26, 27, 31, 32, 35, 38, 41, 42, 43)
copertura base choice (32)
copertura CRUD (34)
copertura dei neuroni (55)
copertura della tabella delle decisioni (37)
copertura dei rami (53)
copertura N-switch (34, 43)
copertura pairwise (32)
correttezza funzionale (44, 45, 62)
criteri di copertura (21, 30, 31, 32, 33, 34, 35)
crowd testing (28, 38, 40)
CRUD (33)
dati anonimizzati (21)
dati di produzione (20)
dati pseudo-anonimizzati (21)
defect arrival pattern (55)
Defect Detection Percentage (54)
Defect Removal Efficiency (51)
diagramma causa-effetto (55)
dominio (30)
dry run (54)
efficacia del contenimento di fase (51)
efficienza nella rimozione dei difetti (51)
elemento della checklist (40)
elemento di copertura (19, 30, 31, 32, 34, 35)
flessibilità (44, 47)
fuzz testing (33)
keyword (13, 21, 22)
identificazione del rischio (24, 25)
ingegneria del caos (33)
installabilità (44, 47)
interoperabilità (44, 48)
lettura perspective-based (50, 54)
livello di rischio (25, 37, 40, 42, 53)
matrice CRUD (33, 34)
matrice di tracciabilità dei requisiti (26)
mitigazione del rischio (24, 25, 26)
modello di sviluppo incrementale (27)
modello di sviluppo iterativo (27)
Modello MBT (Model-Based Testing) (53)
monitoraggio del rischio (24, 26)
N-switch (34, 43, 70)
oracolo umano (20)
partizionamento di equivalenza (30, 42)
partizione di equivalenza (28, 30, 31)
Persona (20, 35, 46, 54)
Phase Containment Effectiveness (51)
portabilità (47)
prevenzione dei difetti (35, 43, 50, 51, 52)
procedura di checksum (37)
profilo operativo (27, 32, 36)
pseudo-oracolo (20)
punto IN (31)
regression testing (24, 26)
relazione metamorfica (28, 37, 38)
retrospettiva (24, 51)
review ad hoc (50, 53)
review individuale (53, 54)
reviewer (53, 54)
ricorrenza dei difetti (54, 55)
rischio di prodotto (24, 25, 41, 47)
selezione dei regression test (26, 27)
semantica dei dati (33, 73)

sostituibilità (47)
sviluppo software Agile (14, 15, 27, 45, 46, 52)
tabella delle decisioni (36, 41, 42)
tabella delle decisioni completa (36)
tabella delle decisioni minimizzata (37)
tassonomia dei difetti (55, 56)
tecnica dei five whys (55, 73)
tecnica di review (50, 53, 54)
tecnica di test basata sui dati (28, 30, 33, 41, 69)
tecnica di test basata sul comportamento (20, 28, 30, 33, 41, 45, 46, 48)
test case di alto livello (13, 15, 16, 18, 37, 38)
test case di basso livello (13, 15, 16, 18, 37, 38)
test case di follow-up (37, 38)
test script automatizzati (17)
testing basato sull'esperienza (16, 28, 38, 40)
testing basato sulla copertura (26)
testing casuale (28, 30, 32, 33, 38, 43)
testing casuale guidato (32)
testing checklist-based (28, 38, 39, 42)
testing combinatorio (28, 30, 31, 43, 47, 52)
testing della tabella delle decisioni (28, 36)
testing di adattabilità (44, 47, 48)
testing di completezza (33)
testing di consistenza (34)
testing CRUD (28, 33, 34)
testing di dominio (28, 31, 42)
testing di flessibilità (44, 47)
testing di interoperabilità (44, 48, 49)
testing di portabilità (47)
testing end-to-end (35)
testing esplorativo (17, 20, 38, 39, 42, 46)
testing funzionale (44, 45, 48)
testing keyword-driven (21, 22)
testing metamorfico (28, 37)
testing model-based (20, 34, 50, 52, 53)
testing property-based (20)
valutazione del rischio (24, 25)