Certificazione di Tester

Syllabus ‘Foundation Level’

Versione 2018 V3.1

International Software Testing Qualifications Board

ITAlian Software Testing Qualifications Board

Contenuto

Storico delle Revisioni

ISTQB® 2018 V3.1	11-November-2019	Release di Manutenzione del Syllabus Certified Tester Foundation Level con modifiche minori – si veda la Release Note
ISTQB® 2018	27-Apr-2018	Candidata alla versione di rilascio
ISTQB® 2011	1-Apr-2011	Release di Manutenzione del Syllabus Certified Tester Foundation Level – si veda la Release Note
ISTQB® 2010	30-Mar-2010	Release di Manutenzione del Syllabus Certified Tester Foundation Level – si veda la Release Note
ISTQB® 2007	01-Mag-2007	Release di Manutenzione del Syllabus Certified Tester Foundation Level
ISTQB® 2005	01-Lug-2005	Syllabus Certified Tester Foundation Level
ASQF V2.2	Lug-2003	ASQF Syllabus Foundation Level Version 2.2 “Lehrplan Grundlagen des Software-testens“
ISEB V2.0	25-Feb-1999	Syllabus ISEB Software Testing Foundation V2.0

Storico delle Revisioni della traduzione ITA-STQB

REV.	AUTORI	DESCRIZIONE DELLE MODIFICHE	DATA DI APPROVAZIONE DI ITA-STQB
01	M.Sogliani	Traduzione del Syllabus	
01	A.Collino – M.Di Carlo	Revisione	25/06/2018
02	C.Sobrero	Revisione in base alla nuova versione Inglese ISTQB® 2018 V3.1	31/03/2020

Ringraziamenti

Questo documento è stato formalmente rilasciato dalla General Assembly di ISTQB® (11 Novembre 2019).

É stato prodotto da un team dedicato appartenente al gruppo di lavoro dell’International Software Testing Qualifications Board: Klaus Olsen (chair), Meile Posthuma e Stephanie Ulrich.

Il team ringrazia i Gruppi di lavoro per i commenti forniti alla review del Syllabus Foundation Level 2018.

Il Syllabus Foundation Level 2018 è stato prodotto da un team dedicato appartenente al gruppo di lavoro dell’International Software Testing Qualifications Board Foundation Level: Klaus Olsen (chair), Tauhida Parveen (vice chair), Rex Black (Project Manager), Debra Friedenberg, Judy McKay, Meile Posthuma, Hans Schaefer, Radoslaw Smilgin, Mike Smith, Steve Toms, Stephanie Ulrich, Marie Walsh, and Eshraka Zakaria.

Il team desidera ringraziare Rex Black and Dorothy Graham per la stesura tecnica, i gruppi di review e tutti i Board nazionali per i suggerimenti e gli input forniti.

Hanno partecipato alla review, al commento e alla votazione di questo Syllabus: Tom Adams, Tobias Ahlgren, Xu Aiguo, Chris Van Bael, Katalin Balla, Graham Bath, Gualtiero Bazzana, Arne Becher, Veronica Belcher, Lars Hilmar Bjørstrup, Ralf Bongard, Armin Born, Robert Bornelind, Mette Bruhn-Pedersen, Geza Bujdoso, Earl Burba, Filipe Carlos, Young Jae Choi, Greg Collina, Alessandro Collino, Cui Zhe, Taz Daughtrey, Matthias Daigl, Wim Decoutere, Frans Dijkman, Klaudia Dussa-Zieger, Yonit Elbaz, Ofer Feldman, Mark Fewster, Florian Fieber, David Frei, Debra Friedenberg, Conrad Fujimoto, Pooja Gautam, Thorsten Geiselhart, Chen Geng, Christian Alexander Graf, Dorothy Graham, Michel Grandjean, Richard Green, Attila Gyuri, Jon Hagar, Kobi Halperin, Matthias Hamburg, Zsolt Hargitai, Satoshi Hasegawa, Berit Hatten, Wang Hongwei, Tamás Horváth, Leanne Howard, Chinthaka Indikadahena, J. Jayapradeep, Kari Kakkonen, Gábor Kapros, Beata Karpinska, Karl Kemminger, Kwanho Kim, Seonjoon Kim, Cecilia Kjellman, Johan Klintin, Corne Kruger, Gerard Kruijff, Peter Kunit, Hyeyong Kwon, Bruno Legeard, Thomas Letzkus, Alon Linetzki, Balder Lingegård, Tilo Linz, Hongbiao Liu, Claire Lohr, Ine Lutterman, Marek Majernik, Rik Marselis, Romanos Matthaios, Judy McKay, Fergus McLachlan, Dénes Medzihradszky, Stefan Merkel, Armin Metzger, Don Mills, Gary Mogyorodi, Ninna Morin, Ingvar Nordström, Adam Novak, Avi Ofer, Magnus C Ohlsson, Joel Oliviera, Monika Stocklein Olsen, Kenji Onishi, Francisca Cano Ortiz, Gitte Ottosen, Tuula Pääkkönen, Ana Paiva, Tal Pe'er, Helmut Pichler, Michaël Pilaeten, Horst Pohlmann, Andrew Pollner, Meile Posthuma, Vitalijs Puiso, Salvatore Reale, Stuart Reid, Ralf Reissing, Shark Ren, Miroslav Renda, Randy Rice, Adam Roman, Jan Sabak, Hans Schaefer, Ina Schieferdecker, Franz Schiller, Jianxiong Shen, Klaus Skafte, Mike Smith, Cristina Sobrero, Marco Sogliani, Murian Song, Emilio Soresi, Helder Sousa, Michael Sowers, Michael Stahl, Lucjan Stapp, Li Suyuan, Toby Thompson, Steve Toms, Sagi Traybel, Sabine Uhde, Stephanie Ulrich, Philippos Vakalakis, Erik van Veenendaal, Marianne Vesterdal, Ernst von Düring, Salinda Wickramasinghe, Marie Walsh, Søren Wassard, Hans Weiberg, Paul Weymouth, Hyungjin Yoon, John Young, Surong Yuan, Ester Zabar, and Karolina Zmitrowicz.

Gruppo di lavoro International Software Testing Qualifications Board per il Syllabus Foundation Level (Edizione 2018): Klaus Olsen (chair), Tauhida Parveen (vice chair), Rex Black (Project Manager), Dani Almog, Debra Friedenberg, Rashed Karim, Johan Klintin, Vipul Kocher, Corne Kruger, Sunny Kwon, Judy McKay, Thomas Müller, Igal Levi, Ebbe Munk, Kenji Onishi, Meile Posthuma, Eric Riou du Cosquer, Hans Schaefer, Radoslaw Smilgin, Mike Smith, Steve Toms, Stephanie Ulrich, Marie Walsh, Eshraka Zakaria, and Stevan Zivanovic. Il Gruppo di lavoro ringrazia il Gruppo di review e tutti i Board nazionali per i suggerimenti e gli input forniti.

Gruppo di lavoro International Software Testing Qualifications Board per il Syllabus Foundation Level (Edizione 2011): Thomas Müller (chair), Debra Friedenberg. Il Gruppo di lavoro ringrazia il Gruppo di revisione (Dan Almog, Armin Beer, Rex Black, Julie Gardiner, Judy McKay, Tuula Pääkkönen, Eric Riou

du Cosquier Hans Schaefer, Stephanie Ulrich, Erik van Veenendaal), e tutti i Board nazionali per i suggerimenti e gli input forniti.

Gruppo di lavoro International Software Testing Qualifications Board per il Syllabus Foundation Level (Edizione 2010): Thomas Müller (chair), Rahul Verma, Martin Klonk and Armin Beer. Il Gruppo di lavoro ringrazia il Gruppo di review (Rex Black, Mette Bruhn-Pederson, Debra Friedenberg, Klaus Olsen, Judy McKay, Tuula

Gruppo di lavoro International Software Testing Qualifications Board per il Syllabus Foundation Level (Edizione 2007): Thomas Müller (chair), Dorothy Graham, Debra Friedenberg, and Erik van Veenendaal. Il Gruppo di lavoro ringrazia il Gruppo di review (Hans Schaefer, Stephanie Ulrich, Meile Posthuma, Anders Pettersson, and Wonil Kwon) e tutti i Board nazionali per i suggerimenti e gli input forniti.

Gruppo di lavoro International Software Testing Qualifications Board per il Syllabus Foundation Level (Edizione 2005): Thomas Müller (chair), Rex Black, Sigrid Eldh, Dorothy Graham, Klaus Olsen, Maaret Pyhäjärvi, Geoff Thompson and Erik van Veenendaal. Il Gruppo di lavoro ringrazia il ruppo di review e tutti i Board nazionali per i suggerimenti e gli input forniti.

Questo documento è stato formalmente rilasciato dall’Assemblea Generale dell’ISTQB® in data 11 Novembre 2019.

0. Introduzione

0.1 Scopo di questo Documento

Questo Syllabus costituisce la base a livello Foundation per l’‘International Software Testing Qualification’.

ISTQB® fornisce questo documento come segue:

1. Ai board nazionali per tradurlo nella loro lingua locale e per accreditare Traininig provider. Tali boards possono adattare il Syllabus alle loro specifiche esigenze linguistiche e aggiungere riferimenti per adattarlo alle pubblicazioni locali.
2. Agli organismi di certificazione per derivare le domande d'esame nella loro lingua locale adattate agli obiettivi di apprendimento del Syllabus stesso.
3. Ai Training Provider, per produrre materiale didattico e determinare i metodi di insegnamento appropriati.
4. Ai candidati alla certificazione, per preparare l'esame di certificazione (come parte di un corso di formazione o autonomamente).
5. Alla comunità internazionale di software e ingegneria dei sistemi, per promuovere la professione del tester di software e di sistemi e come base per libri e articoli.

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

0.2 Il Certified Tester Foundation Level nel Software Testing

La certificazione ‘Certified Tester Foundation Level’ è rivolta a chiunque sia coinvolto nel software testing. Questo include persone che ricoprono ruoli di tester, test analyst, test engineer, test consultant, user acceptance tester e sviluppatori software. Questa certificazione livello Foundation è anche appropriata per chiunque voglia costruirsi una comprensione dei fondamenti del software testing, come Product Owner, Project Manager, Quality Manager, software development manager, Business Analyst, direttori IT e consulenti manageriali.

I possessori del certificato ‘Certified Tester Foundation Level’ saranno in grado di conseguire certificazioni di livello superiore relativamente alle competenze di software testing.

L‘Overview ISTQB® Foundation Level 2018 è un documento separato che è ancora valido per questo Syllabus Foundation Level 2018 v3.1 e include le seguenti informazioni:

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

0.3 Obiettivi di Apprendimento Esaminabili e Livelli di Conoscenza

Gli Obiettivi di Apprendimento supportano i business outcome e vengono utilizzati per creare gli esami Certified Tester Foundation Level.

In generale tutti i contenuti di questo Syllabus sono esaminabili a livello K1, ad eccezione dell'Introduzione e delle Appendici. Al candidato potrebbe quindi essere chiesto di conoscere, ricordare o richiamare una parola chiave o un concetto menzionato in uno qualsiasi dei sei capitoli.

I livelli di conoscenza degli obiettivi di apprendimento, classificati come segue, sono specificati all'inizio di ogni capitolo:

- K1: ricordare
- K2: capire
- K3: applicare

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

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

0.4 L’esame per il Certificato Foundation Level

L'esame per il Certificato Foundation Level 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 e delle Appendici. Standard, libri e altri Syllabi ISTQB® sono inclusi come riferimenti, ma il loro contenuto non è esaminabile, al di là di quanto sintetizzato in questo Syllabus relativamente al loro contenuto.

L’esame è formato da 40 domande a scelta multipla. L’’esame viene superato se almeno il 65% delle domande (cioè 26 domande) ottengono risposte corrette.

Gli esami possono essere svolti come parte di un corso di formazione accreditato o intrapresi indipendentemente. Il completamento di un corso di formazione accreditato non è un prerequisito per l’esame.

0.5 Accreditamento

I Training Provider (Fornitori di Formazione), il cui materiale didattico segue questo Syllabus, possono essere accreditati da un Board nazionale riconosciuto da ISTQB®. Le linee guida per l’accreditamento devono poter essere ottenute dal Board nazionale 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.

0.6 Livello di Dettaglio

Il livello di dettaglio di questo Syllabus consente una formazione ed un esame consistente a livello internazionale. Per raggiungere questo obiettivo, questo Syllabus consiste di:

- Un’indicazione di obiettivi generali che descrivono i propositi del Livello Foundation
- Una lista di termini che lo studente deve ricordare ed avere compreso
- Gli obiettivi di apprendimento per ogni area di conoscenza, che evidenziano il risultato cognitivo che deve essere raggiunto
- Una descrizione dei concetti chiave per l’insegnamento, inclusi i riferimenti a fonti come letteratura universalmente accettata e standard

Il contenuto del Syllabus non è una descrizione dell’intera area di conoscenza del software testing; riflette il livello di dettaglio che deve essere coperto dai corsi per il livello Foundation. Si concentra su concetti di tecniche di test che possono essere applicate a tutti i progetti software, inclusi i progetti Agile. Questo Syllabus non contiene una metodologia o specifici obiettivi di apprendimento relativi a un particolare ciclo di vita dello sviluppo software, ma descrive come questi concetti si applicano nei progetti Agile, in altri tipi di cicli di vita iterativi e incrementali, e nei cicli di vita sequenziali.

0.7 Come è Organizzato questo Syllabus

Sono stati identificati sei capitoli con contenuti esaminabili. Nell’intestazione principale di ogni capitolo è specificato il tempo da dedicare al capitolo stesso; i tempi non sono forniti per i paragrafi interni al capitolo. Per corsi di formazione accreditati, il Syllabus richiede un minimo di 16,75 ore di istruzione, distribuiti nei sei capitoli come segue:

- Capitolo 1: 175 minuti “Fondamenti del Testing”
- Capitolo 2: 100 minuti “Il Testing all’interno del Ciclo di Vita dello Sviluppo Software”
- Capitolo 3: 135 minuti “Testing Statico”
- Capitolo 4: 330 minuti “Tecniche di Test”
- Capitolo 5: 225 minuti “Test Management (Gestione del Test)”
- Capitolo 6: 40 minuti “Strumenti a Supporto del Testing”

1. Fondamenti del Testing – 175 minuti

Parole Chiave

analisi del test, base di test, completamento dei test, condizione di test, controllo dei test, copertura, dati di test, debugging, difetto, errore, esecuzione dei test, failure, implementazione dei test, monitoraggio (monitoring) dei test, obiettivo del test, oggetto di test, oracolo del test, pianificazione dei test, procedura di test, processo di test, progettazione dei test, qualità, quality assurance, root cause, test case (caso di test), test suite, testing, testware, tracciabilità, validazione, verifica

Obiettivi di Apprendimento per i Fondamenti del Testing

1.1 Cos’è il Testing?

FL-1.1.1 (K1) Identificare gli obiettivi tipici del testing

FL-1.1.2 (K2) Differenziare il testing dal debugging

1.2 Perché il Testing è Necessario

FL-1.2.1 (K2) Fornire esempi del perché il testing è necessario

FL-1.2.2 (K2) Descrivere le relazioni fra testing e quality assurance e fornire esempi di come il testing contribuisce al miglioramento della qualità

FL-1.2.3 (K2) Distinguere fra errore, difetto e failure

FL-1.2.4 (K2) Distinguere fra root cause di un difetto e i suoi effetti

1.3 I Sette Principi del Testing

FL-1.3.1 (K2) Spiegare i sette principi del testing

1.4 Il Processo di Test

FL-1.4.1 (K2) Spiegare l’impatto del contesto sul processo di test

FL-1.4.2 (K2) Descrivere le attività di test e i rispettivi compiti nel processo di test

FL-1.4.3 (K2) Differenziare i prodotti di lavoro che supportano il processo di test

FL-1.4.4 (K2) Spiegare il valore di mantenere la tracciabilità fra base di test e prodotti di lavoro del test

1.5 La Psicologia del Testing

FL-1.5.1 (K1) Identificare i fattori psicologici che influenzano il successo del testing

FL-1.5.2 (K2) Spiegare la differenza fra la mentalità richiesta per le attività di test e quella richiesta per le attività di sviluppo

1.1 Cos’è il Testing

I sistemi software sono parte integrante della vita quotidiana, dalle applicazioni aziendali (ad es. bancarie) ai prodotti di consumo (ad es. automobili). La maggior parte delle persone ha avuto esperienze con software che non ha funzionato come previsto. Software che non funziona correttamente può causare molti problemi, tra cui la perdita di denaro, di tempo, di reputazione aziendale e persino infortuni o morte. Il testing del software è un modo per valutare la qualità del software e per ridurre il rischio di failure software in produzione.

Un'errata percezione comune è che il testing consista solo nell’esecuzione dei test, cioè nell'eseguire il software e controllarne i risultati. Come descritto nel paragrafo 1.4, il testing del software è un processo che include molte attività differenti; l'esecuzione dei test (incluso il controllo dei risultati) è solo una di queste attività. Il processo di test include anche attività come la pianificazione, l'analisi, la progettazione e l'implementazione dei test, il reporting dell’avanzamento e dei risultati dei test, e la valutazione della qualità di un oggetto di test.

Alcuni test comportano l'esecuzione del componente o sistema sotto test e tale testing è chiamato testing dinamico. Altri test non richiedono l'esecuzione del componente o sistema da verificare e tale testing è chiamato testing statico. Quindi il testing include anche la review di prodotti di lavoro, come i requisiti, le user story e il codice sorgente.

Un'altra falsa percezione comune sul testing è che esso si concentri interamente sulla verifica dei requisiti, delle user story o di altre specifiche. Il testing comprende anche la verifica che il sistema soddisfi specifici requisiti, cioè che il sistema soddisfi le necessità dell'utente e di altri stakeholder quando posto nel suo ambiente operativo.

Le attività di test sono organizzate e svolte in modo differente nei diversi cicli di vita del software (si veda il paragrafo 2.1).

1.1.1 Obiettivi Tipici del Testing

Per ogni progetto gli obiettivi del testing possono includere:

- Prevenire difetti attraverso la valutazione di prodotti di lavoro come requisiti, user story, progettazione e codice
- Verificare se tutti i requisiti specificati sono stati soddisfatti
- Verificare se l'oggetto di test è completo e validare se funziona come gli utenti e altri stakeholder si aspettano
- Aumentare la confidenza nel livello di qualità dell'oggetto di test
- Trovare failure e difetti, in modo da ridurre il livello di rischio di una qualità del software inadeguata
- Fornire informazioni sufficienti agli stakeholder, per consentire loro di prendere decisioni informate, soprattutto per quanto riguarda il livello di qualità dell'oggetto di test
- Rispettare requisiti o standard contrattuali, legali o normativi, e/o verificare la conformità dell’oggetto di test a tali requisiti o standard

Gli obiettivi del testing possono variare a seconda del contesto del componente o sistema sotto test, del livello di test e del modello del ciclo di vita dello sviluppo software. Queste differenze possono includere, ad esempio:

- Durante il testing di componente, un obiettivo può essere quello di trovare quanti più failure possibile, in modo che i difetti sottostanti siano identificati e corretti in anticipo. Un altro obiettivo potrebbe essere quello di aumentare la copertura del codice.
- Durante il testing di accettazione, un obiettivo può essere quello di confermare che il sistema funzioni come previsto e soddisfi i requisiti. Un altro obiettivo potrebbe essere di fornire agli stakeholder informazioni sul rischio di rilasciare il sistema in un dato momento.

1.1.2 Testing e Debugging

Testing e debugging sono attività differenti. L'esecuzione dei test può rilevare failure causati da difetti nel software. Il debugging è l'attività di sviluppo che trova, analizza e corregge tali difetti. Il successivo testing confermativo verifica se le correzioni abbiano risolto i difetti. In alcuni casi i tester sono responsabili del test iniziale e del test confermativo finale, mentre gli sviluppatori svolgono il debugging, il testing di componente associato e il testing di integrazione dei componenti (continua integrazione). Tuttavia, nello sviluppo Agile e in altri cicli di vita dello sviluppo software, i tester possono essere coinvolti nel debugging e nel testing di componente.

Lo standard ISO (ISO/IEC/IEEE 29119-1) contiene ulteriori informazioni sui concetti di testing del software.

1.2 Perché il Testing è Necessario?

Un testing rigoroso dei componenti e sistemi, inclusa la relativa documentazione associata, può aiutare a ridurre il rischio che si verifichino failure in produzione. Quando i difetti vengono rilevati, e successivamente corretti, questo contribuisce alla qualità dei componenti o dei sistemi. Inoltre, il testing del software può anche essere richiesto per soddisfare requisiti contrattuali o legali, oppure standard specifici in ambito industriale.

1.2.1 Contributi del Testing al Successo

In ambito informatico è abbastanza frequente rilevare situazioni in cui il software e i sistemi siano rilasciati in esercizio e che a valle di tale rilasci, a causa della presenza di difetti, si osservino dei failure o non siano soddisfatte le richieste degli stakeholder. Tuttavia, utilizzando opportune tecniche di test, si è in grado di ridurre la frequenza di tali rilasci problematici, specialmente quando queste tecniche sono applicate con il livello adeguato di competenza nel testing, nei livelli di test appropriati e nei giusti punti del ciclo di vita dello sviluppo software. Ad esempio:

- Avere tester coinvolti nelle review dei requisiti o nel perfezionamento delle user story potrebbe consentire di rilevare difetti in questi prodotti di lavoro. L'identificazione e la correzione dei difetti nei requisiti riduce il rischio di implementare funzionalità errate o non testabili.
- Avere tester che lavorano a stretto contatto con i progettisti di sistema durante la fase di progettazione, può aumentare la reciproca comprensione della progettazione e di come testarla. Questa migliore comprensione può ridurre il rischio di difetti di progettazione e permettere di identificare i test in anticipo.
- Avere tester che lavorano a stretto contatto con gli sviluppatori durante lo sviluppo del codice, può aumentare la reciproca comprensione del codice e di come testarlo. Questa migliore comprensione può ridurre il rischio di introdurre difetti nel codice e nei test.
- Avere tester che verificano e validano il software prima del rilascio può consentire di rilevare failure che potrebbero altrimenti essere trascurati, e supporta il processo di eliminare i difetti che hanno causato i failure (ad es. il debugging). Questo aumenta la probabilità che il software soddisfi i requisiti e le esigenze degli stakeholder.

Oltre a questi esempi, il raggiungimento degli obiettivi di test definiti (si veda il paragrafo 1.1.1) contribuisce al successo complessivo dello sviluppo e della manutenzione del software.

1.2.2 Quality Assurance e Testing

Le persone usano spesso la terminologia *Quality Assurance* (o semplicemente QA) per riferirsi al testing, ma tali aspetti, seppur correlati, non sono la stessa cosa. Un concetto più ampio li lega: il Quality Management (gestione della qualità). Il Quality Management comprende tutte le attività che dirigono e controllano un'organizzazione dal punto di vista della qualità. Tra le altre attività, il Quality Management include sia la Quality Assurance (garanzia della qualità) che il Quality Control (controllo della qualità). La Quality Assurance è tipicamente focalizzata sull'aderenza a processi appropriati, al fine di fornire confidenza sul raggiungimento di livelli adeguati di qualità. Quando i processi vengono eseguiti in modo corretto e appropriato, i prodotti di lavoro creati da questi processi sono generalmente di qualità superiore e ciò contribuisce alla prevenzione dei difetti. Inoltre, l'uso della root cause analysis per rilevare e rimuovere le cause dei difetti, insieme alla corretta gestione dei risultati delle retrospettive per migliorare i processi, sono importanti per un’efficace Quality Assurance.

Il Quality Control coinvolge varie attività, comprese le attività di test, che supportano il raggiungimento di adeguati livelli di qualità. Le attività di test fanno parte del processo generale di sviluppo o di manutenzione del software. Poiché la Quality Assurance riguarda la corretta esecuzione dell'intero processo, questa supporta un testing adeguato. Come descritto nei paragrafi 1.1.1 e 1.2.1, il testing contribuisce al conseguimento della qualità in diversi modi.

1.2.3 Errori, Difetti e Failure

Una persona può commettere un errore che può portare all'introduzione di un difetto (guasto o baco) nel codice software o in un altro prodotto di lavoro correlato. Un errore che porta all'inserimento di un difetto in un prodotto di lavoro può essere il trigger di un altro errore che porta all'inserimento di un difetto in un prodotto di lavoro correlato. Ad esempio, un errore in fase di identificazione dei requisiti può portare a un difetto nei requisiti, con un conseguente errore di programmazione che porterà a un difetto nel codice.

Un difetto nel codice, se eseguito, può causare un failure, ma non necessariamente in tutte le circostanze. Ad es. alcuni difetti richiedono input o precondizioni molto specifici per innescare un failure, che può verificarsi raramente o mai.

Gli errori possono verificarsi per molte ragioni, come ad esempio:

- Vincoli di tempo
- Fallibilità umana
- Partecipanti al progetto inesperti o insufficientemente qualificati
- Errori di comunicazione tra i partecipanti al progetto, che includono problemi di comunicazione relativi ai requisiti e alla progettazione
- Complessità del codice, della progettazione, dell’architettura, del problema da risolvere e/o delle tecnologie utilizzate
- Incomprensioni sulle interfacce interne al sistema e tra sistemi, specialmente quando tali interazioni sono in numero elevato
- Tecnologie nuove o non familiari

Oltre a failure generati da difetti nel codice, i failure possono anche essere causati da condizioni ambientali. Ad esempio, radiazioni, campi elettromagnetici e inquinamento possono causare difetti nel firmware o influenzare l'esecuzione del software modificando le condizioni dell'hardware.

Non tutti i risultati ottenuti differenti dai risultati attesi dei test sono failure. I falsi positivi possono verificarsi a causa di errori nel modo in cui i test sono stati eseguiti o a causa di difetti nei dati di test, nell'ambiente di test o in altri testware, nonché per altri motivi. Si può verificare anche la situazione inversa, dove errori o difetti simili portano a falsi negativi. I falsi negativi sono test che non rilevano i difetti che avrebbero dovuto rilevare; i falsi positivi sono segnalati come difetti, ma non sono in realtà difetti.

1.2.4 Difetti, Root cause ed Effetti

Le root cause dei difetti sono le prime azioni o condizioni che hanno contribuito alla creazione dei difetti. I difetti possono essere analizzati per identificare le loro root cause, in modo da ridurre che si verifichino difetti simili in futuro. Concentrandosi sulle root cause più importanti, la root cause analysis può portare a miglioramenti del processo che impediscano l'introduzione di un numero significativo di difetti futuri.

Ad esempio, si supponga che pagamenti di interessi non corretti, a causa di una singola riga di codice errato, si traducano in lamentele dei clienti. Il codice difettoso è stato scritto per una user story che era ambigua, a causa di un’incomprensione da parte del Product Owner su come calcolare l'interesse. Se esistono molti difetti relativi al calcolo degli interessi, causati da questo tipo di incomprensioni, i Product Owner potrebbero essere istruiti sulla tematica relativa al calcolo degli interessi, per ridurre tali difetti in futuro.

In questo esempio, le lamentele dei clienti sono effetti. I pagamenti di interessi errati sono failure. Il calcolo sbagliato nel codice è un difetto ed è il risultato del difetto originale, cioè l'ambiguità nella user story. La root cause del difetto originale era la mancanza di conoscenza da parte del Product Owner, che lo ha portato a commettere un errore durante la scrittura della user story.

Il processo di root cause analysis è spiegato nel Syllabus ISTQB-CTEL-TM e ISTQB-CTEL-ITP.

1.3 I Sette Principi del Testing

Negli ultimi 50 anni sono stati suggeriti vari principi sul testing che offrono linee guida generali per tutto il testing.

Principio 1 – Il testing mostra la presenza di difetti, non la loro assenza

Il testing può mostrare la presenza di difetti, ma non può provarne l’assenza. Il testing riduce la probabilità che difetti non rilevati rimangano nel software ma, anche se nessun difetto viene trovato, il testing non è una prova di correttezza, ovvero di assenza di difetti.

Principio 2 – Il testing esaustivo è impossibile

Testare tutto (tutte le combinazioni di input e precondizioni) non è fattibile tranne che per casi triviali. Piuttosto che tentare di testare in modo esaustivo, l'analisi del rischio, le tecniche di test e le priorità dovrebbero essere utilizzate per focalizzare l’effort del testing.

Principio 3 – Il testing anticipato permette di risparmiare tempo e denaro

Per trovare i difetti in anticipo, è necessario iniziare attività di test statico e dinamico il prima possibile nel ciclo di vita dello sviluppo software. Il testing anticipato è a volte indicato come “*shift left”*. Eseguire il test all'inizio del ciclo di vita dello sviluppo software aiuta a ridurre o eliminare le costose modifiche successive (si veda il paragrafo 3.1).

Principio 4 – I difetti tendono a formare cluster

Un piccolo numero di moduli solitamente contiene la maggior parte dei difetti scoperti durante i test prima del rilascio, o è responsabile della maggior parte dei failure in esercizio. I cluster dei difetti previsti e quelli effettivamente osservati in fase di test o in esercizio sono un input importante a un'analisi del rischio utilizzata per focalizzare l’effort del testing (come riportato nel principio 2).

Principio 5 – Attenzione al paradosso pesticida

Se gli stessi test vengono ripetuti più e più volte, presumibilmente tali test non rileveranno ulteriori nuovi difetti. Per rilevare nuovi difetti, i test esistenti e i dati di test potrebbero dover essere modificati, e potrebbe essere necessario progettare nuovi test (i test non sono più efficaci nel rilevare difetti, proprio come i pesticidi non risultano più efficaci nell'uccidere gli insetti dopo un utilizzo protratto nel tempo). In alcuni casi, come nei regression test automatizzati, il paradosso pesticida ha un risultato favorevole, che consiste in un numero relativamente basso di difetti di regressione.

Principio 6 – Il testing è dipendente dal contesto

Il testing viene eseguito in modo differente in contesti differenti. Ad esempio, Il test di un software di controllo industriale safety-critical è differente da quello svolto per una app mobile di e-commerce. Come altro esempio, il testing in un progetto Agile viene svolto in modo diverso rispetto al testing in un progetto con ciclo di vita dello sviluppo software sequenziale (si veda il paragrafo 2.1).

Principio 7 – L’assenza di errori è una falsa credenza

Alcune organizzazioni si aspettano che i tester possano eseguire tutti i test possibili e trovare tutti i difetti possibili, ma i principi 2 e 1 ci dicono, rispettivamente, che questo è impossibile. Inoltre, è una falsa credenza aspettarsi che il solo trovare e risolvere un gran numero di difetti garantisca il successo di un sistema. Ad esempio, eseguire il testing accurato di tutti i requisiti specificati e correggere tutti i difetti rilevati potrebbe comunque non essere sufficiente ad evitare che venga rilasciato un sistema difficile da usare, che non soddisfi le necessità e le aspettative degli utenti, o che risulti inferiore rispetto ad altri sistemi competitivi.

Si veda Myers 2011, Kaner 2002, Weinberg 2008 e Beizer 1990 per esempi di questi e altri principi sul testing.

1.4 Il Processo di Test

Non esiste un unico processo universale di test del software, ma esistono insiemi comuni di attività di test senza le quali il testing avrà meno probabilità di raggiungere gli obiettivi stabiliti. Questi insiemi di attività di test costituiscono un processo di test. Un adeguato e specifico processo di test del software per ogni situazione dipende da molti fattori. Quali attività di test sono coinvolte in questo processo, come queste attività sono implementate e quando queste attività possono essere pianificate possono essere discusse in una strategia di test dell’organizzazione.

1.4.1 Processo di Test nel Contesto

I fattori contestuali che influenzano il processo di test di un'organizzazione includono tra gli altri:

- Il modello di ciclo di vita dello sviluppo software e le metodologie di progetto utilizzate
- I livelli di test e i tipi di test considerati
- I rischi di prodotto e di progetto
- Il dominio di business
- I vincoli operativi, tra cui:
  - Budget e risorse
  - Tempistiche
  - Complessità
  - Requisiti contrattuali e normativi
- Le politiche e le pratiche organizzative
- Gli standard interni ed esterni richiesti

I seguenti paragrafi descrivono aspetti generali dell’organizzazione dei processi di test in termini di:

- Attività e compiti del test
- Prodotti di lavoro del test
- Tracciabilità tra base di test e prodotti di lavoro del test

È molto utile che la base di test (qualsiasi sia il livello o tipo di test che si stia considerando) abbia criteri di copertura definiti e misurabili. I criteri di copertura, come Key Performance Indicator (KPI), possono agire efficacemente per guidare le attività finalizzate a dimostrare il raggiungimento degli obiettivi di test del software (si veda il paragrafo 1.1.1).

Ad esempio, per un'applicazione mobile, la base di test può includere un elenco di requisiti e un elenco di dispositivi mobili supportati. Ogni requisito è un elemento della base di test. Ogni dispositivo supportato è anch’esso un elemento della base di test. I criteri di copertura possono richiedere almeno un test case per ciascun elemento della base di test. Una volta eseguiti, i risultati di questi test indicano agli stakeholder se i requisiti specificati sono soddisfatti e se sono stati rilevati failure sui dispositivi supportati.

Lo standard ISO (ISO/IEC/IEEE 29119-2) contiene ulteriori informazioni sui processi di test.

1.4.2 Attività e Compiti del Test

Un processo di test è costituito dai seguenti gruppi principali di attività:

- Pianificazione dei test
- Monitoraggio e controllo dei test
- Analisi dei test
- Progettazione dei test
- Implementazione dei test
- Esecuzione dei test
- Completamento dei test

Ogni gruppo principale di attività è composto da attività costituenti che saranno descritte nei prossimi paragrafi. Ogni attività constituente consiste di più attività individuali, che possono variare da un progetto o da un rilascio all'altro.

Inoltre, sebbene molti di questi gruppi principali di attività possono apparire logicamente sequenziali, sono spesso implementati iterativamente. Ad esempio, lo sviluppo Agile è strutturato in piccole iterazioni di progettazione, sviluppo e test del software che sono svolte in modo continuativo e supportate da una pianificazione continua. Perciò le attività di test sono anch’esse svolte su base iterativa e continuativa all'interno di questo approccio di sviluppo software. Anche nello sviluppo software sequenziale, la sequenza logica dei gruppi principali di attività può comprendere sovrapposizioni, combinazioni, concorrenza oppure omissioni, per cui è generalmente richiesto di eseguire un tailoring di questi gruppi principali di attività nel contesto del sistema e del progetto.

Pianificazione dei test

La pianificazione dei test comprende attività che definiscono gli obiettivi del test e l'approccio per soddisfare tali obiettivi, nell’ambito dei vincoli imposti dal contesto (ad es. specificando tecniche e compiti di test idonei e prevedendo una schedulazione dei test per rispettare una scadenza). I Test Plan (Piani di Test) possono essere modificati in base al feedback dalle attività di monitoraggio e controllo. La pianificazione dei test è ulteriormente spiegata nel paragrafo 5.2.

Monitoraggio e controllo dei test

Il monitoraggio dei test prevede il confronto continuo dell’avanzamento effettivo rispetto all’avanzamento pianificato, utilizzando le metriche di monitoraggio dei test definite nel Test Plan (Piano di Test). Il controllo dei test prevede l'implementazione delle azioni necessarie a soddisfare gli obiettivi del Test Plan (che può essere aggiornato nel tempo). Il monitoraggio e il controllo dei test sono supportati dalla valutazione dei criteri di uscita, che sono indicati come “Definition of Done” in alcuni cicli di vita dello sviluppo software(si veda ISTQB-CTFL-AT). Ad esempio, la valutazione dei criteri di uscita per l'esecuzione dei test nell’ambito di un determinato livello di test può includere:

- Verifica dei risultati dei test e dei test log rispetto ai criteri di copertura specificati
- Valutazione del livello di qualità del componente o sistema in base ai risultati del test e ai test log

- Valutazione della necessità di ulteriori test (ad es. se i test originariamente progettati non hanno permesso di raggiungere il livello di copertura del rischio di prodotto desiderato, richiedendo test aggiuntivi da progettare ed eseguire)

L’avanzamento dei test rispetto al pianificato è comunicato agli stakeholder nei Test Progress Report, includendo le deviazioni dal piano e le informazioni per supportare qualsiasi decisione di interrompere il testing.

Il monitoraggio e il controllo dei test sono ulteriormente spiegati nel paragrafo 5.3.

Analisi dei test

Durante l'analisi dei test, la base di test viene analizzata per identificare le funzionalità testabili e per definire le condizioni di test associate. In altre parole l'analisi dei test determina "cosa testare" in termini di criteri di copertura misurabili.

L'analisi dei test include le seguenti attività principali:

- Analizzare la base di test adeguata al livello di test considerato, ad esempio:
  - Specifiche dei requisiti, come requisiti di business, requisiti funzionali, requisiti di sistema, user story, epic, use case o prodotti di lavoro simili che specificano il comportamento funzionale e non-funzionale desiderato di componenti o sistemi
  - Informazioni sulla progettazione e implementazione, come l’architettura di sistema o del software, diagrammi o documenti, specifiche di progettazione, grafi di flussi di chiamata, diagrammi di modellazione (ad es. diagrammi UML o entity-relationship diagram), specifiche di interfaccia o prodotti di lavoro simili che descrivano la struttura del componente o del sistema
  - Implementazione del componente o del sistema stesso, inclusi codice, interfacce, metadati e query del database
  - Report di analisi del rischio, che possono considerare aspetti funzionali, non-funzionali e strutturali del componente o del sistema
- Valutare la base di test e gli elementi di test per identificare difetti di vario tipo, come ad esempio:
  - Ambiguità
  - Omissioni
  - Inconsistenze
  - Inaccuratezze
  - Contraddizioni
  - Dichiarazioni superflue
- Identificare le funzionalità e gli insiemi di funzionalità da testare
- Definire ed assegnare le priorità alle condizioni di test per ciascuna funzionalità, in base all'analisi della base di test e considerando le caratteristiche funzionali, non-funzionali e strutturali, cosi come altri fattori di business e tecnici, e i livelli di rischio
- Creare la tracciabilità bidirezionale tra ciascun elemento della base di test e le condizioni di test associate (si vedano i paragrafi 1.4.3 e 1.4.4)

L'applicazione di tecniche di test black-box, white-box e basate sull'esperienza può essere utile nel processo di analisi dei test (si veda il capitolo 4) per ridurre la probabilità di omettere condizioni di test importanti e per definire condizioni di test più precise e accurate.

In alcuni casi, l'analisi dei test produce condizioni di test che devono essere usate come obiettivi di test nei Test Charter. I Test Charter sono prodotti di lavoro tipici in alcuni tipi di test basati sull'esperienza (si veda il paragrafo 4.4.2). Quando questi obiettivi di test sono tracciabili rispetto alla base di test, è possibile misurare la copertura raggiunta da tali test basati sull'esperienza.

L'identificazione di difetti durante l'analisi dei test è un importante vantaggio potenziale, specialmente dove non venga utilizzato nessun altro processo di review e/o il processo di test sia strettamente connesso con il processo di review. Tali attività di analisi dei test non solo verificano che i requisiti siano consistenti, espressi appropriatamente e completi, ma validano anche che i requisiti catturino correttamente le esigenze del cliente, dell’utente e degli altri stakeholder. Ad esempio, Le tecniche come Behavior Driven Development (BDD) e Acceptance Test Driven Development (ATDD), generano le condizioni di test e i test case dalle user story e dai criteri di accettazione prima della codifica. Queste tecniche, possono anche verificare, validare e rilevare difetti nelle user story e nei criteri di accettazione associati (si veda ISTQB-CTFL-AT).

Progettazione dei test

Durante la progettazione dei test, le condizioni di test vengono elaborate in test case di alto livello, insiemi di test case di alto livello e altro testware. Quindi, l'analisi dei test risponde alla domanda "cosa testare?", mentre la progettazione dei test risponde alla domanda "come testare?"

La progettazione dei test include le seguenti attività principali:

- Progettare e prioritizzare i test case e gli insiemi di test case
- Identificare i dati di test necessari a supportare le condizioni di test e i test case
- Progettare l'ambiente di test e identificare l’infrastruttura e gli strumenti richiesti
- Creare la tracciabilità bidirezionale tra base di test, condizioni di test e test case (si veda il paragrafo 1.4.4)

L'elaborazione delle condizioni di test in test case e insiemi di test case durante la progettazione dei test spesso comporta l’utilizzo di tecniche di test (si veda il capitolo 4).

Come nell’analisi dei test, anche la progettazione dei test può identificare tipi di difetti similari nella base di test. Analogamente all’analisi dei test, l’identificazione di difetti durante la progettazione dei test è un importante beneficio potenziale.

Implementazione dei test

Durante l'implementazione dei test, viene creato e/o completato il testware necessario per l'esecuzione, compresa la sequenzializzazione dei test case in procedure di test. Quindi, la progettazione dei test risponde alla domanda "come testare?", mentre l'implementazione dei test risponde alla domanda "è tutto correttamente predisposto per garantire l’esecuzione dei test?”

L'implementazione dei test include le seguenti attività principali:

- Sviluppare e prioritizzare le procedure di test e, potenzialmente, creare i test script automatizzati
- Creare le test suite dalle procedure di test e (se presenti) da test script automatizzati
- Organizzare le test suite all'interno di una schedulazione dell'esecuzione dei test, in modo da ottenere un’esecuzione efficiente dei test (si veda il paragrafo 5.2.4)
- Creare l'ambiente di test (compresi, potenzialmente, test harness, virtualizzazione dei servizi, simulatori e altri elementi dell'infrastruttura) e verificare che tutto il necessario sia stato correttamente predisposto
- Preparare i dati di test e assicurare che siano correttamente caricati nell'ambiente di test
- Verificare e aggiornare la tracciabilità bidirezionale tra base di test, condizioni di test, test case, procedure di test e test suite (si veda il paragrafo 1.4.4)

Le attività di progettazione e implementazione dei test sono spesso combinate.

Nel testing esplorativo e in altre tecniche basate sull'esperienza, sia la progettazione che l’implementazione dei test possono essere svolte e documentate come parte dell'esecuzione dei test. Il testing esplorativo può essere basato su Test Charter (prodotti come parte dell'analisi dei test) e i test

esplorativi vengono eseguiti nello stesso momento in cui sono progettati e implementati (si veda il paragrafo 4.4.2).

Esecuzione dei test

Durante l'esecuzione dei test, le test suite vengono eseguite in base alla schedulazione dell'esecuzione dei test.

L'esecuzione dei test include le seguenti attività principali:

- Registrare gli ID e le versioni degli elementi di test o dell'oggetto di test, degli strumenti di test e del testware
- Eseguire i test manualmente o tramite strumenti di esecuzione dei test (Test Automation)
- Confrontare i risultati effettivi con i risultati attesi
- Analizzare le anomalie per stabilire le loro cause probabili (ad es. possono verificarsi failure dovuti a difetti nel codice, ma possono verificarsi anche falsi positivi (si veda il paragrafo 1.2.3)
- Segnalare i difetti in base ai failure osservati (si veda il paragrafo 5.6)
- Memorizzare i risultati dell'esecuzione dei test (ad es. passato, fallito, bloccato)
- Ripetere le attività di test a seguito di azioni intraprese per un'anomalia o come parte del testing pianificato (ad es. esecuzione di un test corretto, testing confermativo e/o regression testing)
- Verificare e aggiornare la tracciabilità bidirezionale tra base di test, condizioni di test, test case, procedure di test e risultati dei test

Completamento dei test

Le attività di completamento dei test raccolgono dati dalle attività di test completate per consolidare l’esperienza, il testware e ogni altra informazione pertinente. Tali attività vengono svolte al raggiungimento di milestone predefinite del progetto, come al rilascio di un sistema software, al completamento o alla cancellazione di un progetto di test, al termine di un'iterazione di un progetto Agile, al completamento di un livello di test o di unrilascio di manutenzione.

Il completamento dei test include le seguenti attività principali:

- Verificare la chiusura di tutti i defect report, creare change request (richieste di modifica) o elementi nel Product Backlog per eventuali difetti rimasti irrisolti alla fine dell'esecuzione dei test
- Creare un Test Summary Report da comunicare agli stakeholder
- Finalizzare e archiviare l'ambiente di test, i dati di test, l'infrastruttura di test e altro testware per il successivo riutilizzo
- Consegnare il testware ai gruppi di manutenzione, ad altri gruppi di progetto e/o ad altri stakeholder che potrebbero beneficiare del loro utilizzo
- Analizzare le “lessons learned” dalle attività di test completate, per determinare le modifiche necessarie per future iterazioni, rilasci e progetti
- Utilizzare le informazioni raccolte per migliorare la maturità del processo di test

1.4.3 Prodotti di Lavoro del Test

I prodotti di lavoro del test vengono creati come parte del processo di test. Proprio come esiste una variabilità significativa nel modo con cui le organizzazioni implementano il processo di test, esiste anche una significativa variabilità nei tipi prodotti di lavoro creati durante tale processo, nel modo in cui i prodotti di lavoro sono organizzati e gestiti e nei nomi ad essi attribuiti. Questo Syllabus aderisce al processo di test sopra descritto, i cui prodotti di lavoro sono descritti in questo Syllabus e nel glossario ISTQB®. Lo standard ISO (ISO/IEC/IEEE 29119-3) può anch’esso essere considerato una linea guida per i prodotti di lavoro del test.

Molti dei prodotti di lavoro del test descritti in questo paragrafo possono essere creati e gestiti utilizzando strumenti di Test Management (gestione del test) e strumenti di Defect Management (gestione dei difetti) (si veda il capitolo 6).

Prodotti di lavoro della pianificazione dei test

I prodotti di lavoro della pianificazione dei test includono in genere uno o più Test Plan. Il Test Plan include informazioni sulla base di test, a cui gli altri prodotti di lavoro di test saranno correlati attraverso informazioni sulla tracciabilità (si veda di seguito e il paragrafo 1.4.4), nonché i criteri di uscita (o Definition of Done) che verranno utilizzati durante il monitoraggio e controllo dei test. I Test Plan sono descritti nel paragrafo 5.2.

Prodotti di lavoro del monitoraggio e controllo dei test

I prodotti di lavoro del monitoraggio e controllo dei test di solito includono vari tipi di test report, inclusi i Test Progress Report redatti su base continuativa e/o periodica, e i Test Summary Report, redatti per le diverse milestone di completamento. Tutti i test report dovrebbero fornire dettagli significativi ai destinatari in termini di avanzamento dei test alla data dei report stessi, includendo il riepilogo dei risultati dell'esecuzione dei test una volta che questi diventino disponibili.

I prodotti di lavoro del monitoraggio e controllo dei test dovrebbero anche indirizzare i problemi di Project Management (gestione del progetto), come ad esempio il completamento delle attività, l’allocazione e l’utilizzo delle risorse, cosi come l’effort associato.

I prodotti di lavoro creati durante queste attività sono ulteriormente descritti nel paragrafo 5.3.

Prodotti di lavoro dell’analisi dei test

I prodotti di lavoro dell’analisi dei test includono le condizioni di test definite e prioritizzate, ognuna delle quali è idealmente tracciabile in modo bidirezionale allo specifico elemento della base di test coperto da tale condizione. Per il testing esplorativo, l'analisi può comportare la creazione dei Test Charter. L'analisi dei test può portare anche alla scoperta e segnalazione di difetti nella base di test.

Prodotti di lavoro della progettazione dei test

I prodotti di lavoro della progettazione dei test includono i test case e gli insiemi di test case necessari ad esercitare le condizioni di test definite durantel’analisi dei test. Spesso è una buona pratica progettare test case di alto livello, senza valori concreti per i dati di input e i risultati attesi. Tali test case di alto livello sono riutilizzabili in più cicli di test con diversi dati concreti, documentando ancora in modo appropriato l’ambito del test case. Idealmente ogni test case è tracciabile in modo bidirezionale alle condizioni di test coperte da tale test case.

La progettazione dei test comporta anche:

- La progettazione e/o l'identificazione dei dati di test necessari
- La progettazione dell’ambiente di test
- L'identificazione di infrastrutture e strumenti

La misura in cui questi risultati sono documentati può variare significativamente.

Prodotti di lavoro dell’implementazione dei test

I prodotti di lavoro dell’implementazione dei test includono:

- Procedure di test e sequenza di tali procedure
- Test suite
- Una schedulazione dell'esecuzione dei test

Idealmente, una volta completata l'implementazione dei test, il raggiungimento dei criteri di copertura stabiliti nel Test Plan può essere dimostrato attraverso la tracciabilità bidirezionale tra le procedure di test e gli elementi specifici della base di test, attraverso i test case e le condizioni di test.

In alcuni casi, l'implementazione dei test comporta la creazione di prodotti di lavoro che utilizzano o sono utilizzati da strumenti, come la virtualizzazione dei servizi e i test script automatizzati.

L'implementazione dei test può anche comportare la creazione e la verifica dei dati di test e dell'ambiente di test. La completezza della documentazione dei dati e/o dei risultati della verifica dell’ambiente possono variare in modo significativo.

I dati di test servono per assegnare valori concreti agli input e ai risultati attesi dei test case. Tali valori concreti, insieme alle indicazioni esplicite sul loro uso, trasformano i test case di alto livello in test case di basso livello eseguibili. Lo stesso test case di alto livello può utilizzare dati di test diversi, quando eseguito su versioni diverse dell'oggetto di test. I risultati attesi concreti, che sono associati ai dati di test concreti, vengono identificati utilizzando un oracolo di test.

Nel testing esplorativo, alcuni prodotti di lavoro della progettazione e dell’implementazione dei test possono essere creati durante l’esecuzione dei test, anche se la misura con cui i test esplorativi (e la loro tracciabilità agli elementi specifici della base di test) vengono documentati può variare in modo significativo.

Le condizioni di test definite nell'analisi dei test possono essere ulteriormente raffinate nell’implementazione dei test.

Prodotti di lavoro dell’esecuzione dei test

I prodotti di lavoro dell’esecuzione dei test includono:

- Documentazione sullo stato dei singoli test case o delle procedure di test (ad es. pronto per essere eseguito, superato, fallito, bloccato, non eseguito volutamente, ecc.)
- Defect report (si veda il paragrafo 5.6)
- Documentazione su quali elementi di test, oggetti di test, strumenti di test e testware sono stati coinvolti nel testing

Idealmente, una volta completata l'esecuzione dei test, lo stato di ciascun elemento della base di test può essere determinato e segnalato tramite la tracciabilità bidirezionale con le procedure di test associate. Ad esempio, è possibile dire quali requisiti hanno superato tutti i test pianificati, quali requisiti hanno dei test falliti e/o dei difetti ad essi associati, e quali requisiti hanno dei test pianificati ancora in attesa di essere eseguiti. Questo consente di verificare se i criteri di copertura sono stati soddisfatti e permette di riportare i risultati dei test in modo comprensibile agli stakeholder.

Prodotti di lavoro del completamento dei test

I prodotti di lavoro del completamento dei test includono i Test Summary Report, le azioni di miglioramento per i successivi progetti o iterazioni, le change request o gli elementi del product backlog, e il testware finalizzato.

1.4.4 Tracciabilità tra Base di Test e Prodotti di Lavoro del Test

Come menzionato nel paragrafo 1.4.3, i prodotti di lavoro del testing e i loro nomi variano in modo significativo. Indipendentemente da queste variazioni, al fine di implementare un monitoraggio e un controllo dei test che siano efficaci, è importante, durante il processo di test, stabilire e mantenere la tracciabilità tra ciascun elemento della base di test e i vari prodotti di lavoro associati a tale elemento, come descritto sopra. Oltre alla valutazione della copertura dei test, una buona tracciabilità supporta:

- Analisi degli impatti delle modifiche (change impact analysis)
- Possibilità di eseguire attività di audit al testing
- Conformità ai criteri di governance IT
- Miglioramento della comprensibilità dei Test Progress Report e dei Test Summary Report includendo lo stato degli elementi della base di test (ad es. i requisiti che hanno superato tutti i test, i requisiti per cui sono falliti i test e i requisiti che hanno test in sospeso)
- Correlazione degli aspetti tecnici del test a termini comprensibili agli stakeholder
- Disponibilità di informazioni per valutare la qualità del prodotto, l’adeguatezza del processo e lo stato di avanzamento del progetto rispetto agli obiettivi di business

Alcuni strumenti di Test Management forniscono modelli dei prodotti di lavoro del testing che corrispondono totalmente o parzialmente ai prodotti di lavoro descritti in questo paragrafo. Alcune organizzazioni implementano i propri sistemi di gestione per organizzare i prodotti di lavoro e fornire la tracciabilità delle informazioni di cui hanno bisogno.

1.5 La Psicologia del Testing

Lo sviluppo del software, compreso il testing del software, coinvolge esseri umani. Pertanto, la psicologia umana ha effetti importanti sul testing del software.

1.5.1 Psicologia Umana e Testing

L’identificazione dei difetti durante un testing statico come una review dei requisiti o una sessione di raffinamento delle user story, oppure il rilevamento dei failure durante l'esecuzione del testing dinamico, possono essere percepiti come critiche al prodotto e al suo autore. Un elemento della psicologia umana chiamato “confirmation bias” (pregiudizio di conferma) può rendere difficile accettare informazioni che non concordino con i propri principi e le proprie idee. Ad esempio, poiché gli sviluppatori sono convinti che il proprio codice sia corretto, hanno un confirmation bias che rende loro difficile accettare che il codice sia errato. In aggiunta ai confirmation bias, i “cognitive bias” (pregiudizi cognitivi) possono rendere difficile capire o accettare le informazioni prodotte dai test. Infine, è una comune caratteristica umana dare la colpa a colui che viene percepito come “portatore di cattive notizie”, e le informazioni prodotte dal testing contengono spesso cattive notizie.

Come risultato di questi fattori psicologici, alcune persone possono percepire il testing come un'attività distruttiva, anche se contribuisce enormemente all’avanzamento del progetto e alla qualità del prodotto (si vedano i paragrafi 1.1 e 1.2). Per cercare di ridurre queste percezioni, le informazioni su difetti e failure dovrebbero essere comunicate in modo costruttivo. In questo modo, possono essere ridotte le tensioni tra i tester e gli analisti, i Product Owner, i progettisti e gli sviluppatori. Questo si applica sia per il testing statico che per il testing dinamico.

I tester e i Test Manager devono possedere buone capacità interpersonali per essere in grado di comunicare in modo efficace i difetti, i failure, i risultati dei test, gli avanzamenti dei test e i rischi, e per costruire relazioni positive con i colleghi. Esempi di modalità per instaurare una comunicazione positiva includono:

- Iniziare a collaborare invece di combattere. Ricordare a tutti l'obiettivo comune sistemi di qualità migliori.
- Enfatizzare i benefici del testing. Ad esempio, le informazioni sui difetti possono aiutare gli autori a migliorare i loro prodotti di lavoro e le loro capacità. I difetti rilevati e corretti durante il testing consentiranno all’organizzazione di risparmiare tempo e denaro e ridurre il rischio complessivo sulla qualità del prodotto.
    - Comunicare i risultati dei test e altre evidenze in modo neutrale, focalizzato sui fatti, senza criticare la persona che ha creato l'elemento difettoso. Scrivere defect report oggettivi e basati sui fatti, e svolgere una review dei risultati.
- Cercare di comprendere i sentimenti dell'altra persona e le ragioni per cui può reagire negativamente all’informazione ricevuta.
- Ottenere conferma che entrambe le parti abbiano capito ciò che è stato detto.

Gli obiettivi tipici del test sono già stati discussi (si veda il paragrafo 1.1). La chiara definizione del corretto insieme di obiettivi del test ha importanti implicazioni psicologiche. La maggior parte delle persone tende ad allineare i propri piani e comportamenti agli obiettivi stabiliti dal team, dal management e dagli altri stakeholder. È inoltre importante che i tester aderiscano a questi obiettivi col minimo pregiudizio personale.

1.5.2 Mentalità di Tester e Sviluppatori

Tester e sviluppatori spesso pensano in modo diverso. L’obiettivo principale dello sviluppo è progettare e costruire un prodotto. Come discusso in precedenza, gli obiettivi del test includono la verifica e la validazione del prodotto, la rilevazione di difetti prima del rilascio e cosi via. Si tratta quindi di differenti insiemi di obiettivi che richiedono diverse mentalità. Tenere assieme queste diverse mentalità consente di ottenere un livello di qualità del prodotto più elevato.

Una mentalità riflette le assunzioni di un individuo e i metodi preferiti per prendere decisioni e risolvere problemi. La mentalità di un tester dovrebbe includere curiosità, pessimismo professionale, occhio critico, attenzione al dettaglio e predisposizione a comunicazioni e relazioni buone e positive. La mentalità di un tester tende crescere e maturare con l’esperienza.

La mentalità di uno sviluppatore può includere alcuni degli elementi della mentalità di un tester, ma sviluppatori di successo sono spesso più interessati a progettare e costruire soluzioni piuttosto che a identificare ciò che potrebbe essere sbagliato in tali soluzioni. Inoltre, il confirmation bias rende difficile prendere coscienza dei propri errori.

Con la giusta mentalità, gli sviluppatori sono in grado di testare il proprio codice. I diversi modelli del ciclo di vita dello sviluppo software hanno spesso diversi modi di organizzare i tester e le attività di test. Quando alcune delle attività di test sono svolte da tester indipendenti, questo può aumentare l'efficacia del rilevamento dei difetti, il che è particolarmente importante per sistemi di grandi dimensioni, complessi o safety-critical. I tester indipendenti hanno un punto di vista differente da quello degli autori del prodotto di lavoro (ad esempio business analyst, Product Owner, progettisti e sviluppatori), poiché hanno cognitive bias diversi dagli autori.

2. Il Testing all’interno del Ciclo di Vita dello Sviluppo Software – 100 minuti

Parole Chiave

alpha testing, ambiente di test, analisi degli impatti, base di test, beta testing, COTS (Commercial Off-The-Shelf), livello di test, modello di sviluppo sequenziale, obiettivo del test, oggetto di test, regression testing, test case (caso di test), testing basato sulle modifiche, testing di accettazione, testing di accettazione contrattuale, testing di accettazione normativo, testing di accettazione operativo, testing di accettazione utente, testing di componente, testing confermativo, testing funzionale, testing di integrazione, testing di integrazione dei componenti, testing di integrazione dei sistemi, testing di manutenzione, testing di sistema, testing non-funzionale, testing relativo a modifiche, testing white-box, tipo di test

Obiettivi di Apprendimento per il Testing all’interno del Ciclo di Vita dello Sviluppo Software

2.1 Modelli del Ciclo di Vita dello Sviluppo Software

FL-2.1.1 (K2) Spiegare le relazioni tra attività di sviluppo software e attività di test nel ciclo di vita dello sviluppo software

FL-2.1.2 (K1) Identificare le ragioni per cui i modelli del ciclo di vita dello sviluppo software devono essere adattati al contesto del progetto e alle caratteristiche del prodotto

2.2 Livelli di Test

FL-2.2.1 (K2) Confrontare i diversi livelli di test dal punto di vista di obiettivi, basi di test, oggetti di test, difetti e failure tipici, approcci e responsabilità

2.3 Tipi di Test

FL-2.3.1 (K2) Confrontare il testing funzionale, il testing non-funzionale e il testing white-box

FL-2.3.2 (K1) Riconoscere che i test funzionali, non funzionali e white-box si svolgono a qualsiasi livello di test

FL-2.3.3 (K2) Confrontare lo scopo del testing confermativo e del regression testing

2.4 Testing di Manutenzione

FL-2.4.1 (K2) Riepilogare i trigger per il testing di manutenzione

FL-2.4.2 (K2) Descrivere il ruolo dell'analisi degli impatti nel testing di manutenzione

2.1 Modelli del Ciclo di Vita dello Sviluppo Software

Un modello del ciclo di vita dello sviluppo software descrive i tipi di attività eseguiti in ogni fase di un progetto di sviluppo software, e come le attività si relazionano tra loro in modo logico e cronologico. Esistono diversi modelli del ciclo di vita dello sviluppo software, ognuno dei quali richiede diversi approcci al testing.

2.1.1 Sviluppo del Software e Testing del Software

Una parte importante del ruolo di un tester è di avere familiarità con i modelli del ciclo di vita dello sviluppo software, in modo da poter essere svolte appropriate attività di test .

In ogni modello del ciclo di vita dello sviluppo software, possono essere applicate buone pratiche di testing:

- Per ogni attività di sviluppo, esiste una corrispondente attività di test
- Ogni livello di test ha obiettivi del test specifici per quel livello
- L'analisi e la progettazione dei test per un determinato livello di test iniziano durante la corrispondente attività di sviluppo
- I tester partecipano alle discussioni per definire e raffinare i requisiti e la progettazione, e sono coinvolti nella review dei prodotti di lavoro (ad esempio requisiti, progettazione, user story, ecc.) non appena le versioni draft sono disponibili

Indipendentemente dalla scelta del modello del ciclo di vita dello sviluppo software, le attività di test dovrebbero iniziare nelle prime fasi del ciclo di vita, sulla base del principio di testing anticipato.

Questo Syllabus classifica i tradizionali modelli del ciclo di vita dello sviluppo del software come segue:

- Modelli di sviluppo sequenziali
- Modelli di sviluppo iterativi e incrementali

Un modello di sviluppo sequenziale descrive il processo di sviluppo del software come un flusso lineare e sequenziale di attività. Questo significa che qualsiasi fase del processo di sviluppo dovrebbe iniziare quando la precedente fase è stata completata. In teoria non esistono sovrapposizioni di fasi, ma nella pratica è utile avere feedback anticipati dalla fase successiva.

Nel modello a cascata (Waterfall), le attività di sviluppo (ad esempio analisi dei requisiti, progettazione, codifica, testing) sono completate in sequenza, una dietro l’altra. In questo modello le attività di test sono quando tutte le altre attività di sviluppo sono state completate.

A differenza del modello Waterfall, il V-Model implementa il principio del testing anticipato, integrando il processo di test nel processo di sviluppo, e associando un livello di test alla corrispondente fase di sviluppo (si veda il paragrafo 2.2 per la descrizione dei livelli di test). In questo modello l'esecuzione dei test associati a un livello di test procede sequenzialmente, ma in alcuni casi i livelli possono sovrapporsi.

I modelli di sviluppo sequenziali rilasciano software che contiene l’insieme completo di funzionalità, ma in genere richiedono mesi o anni per il rilascio agli stakeholder e agli utenti.

Nel modello di sviluppo incrementale le attività di definizione dei requisiti, progettazione, sviluppo e testing di un sistema vengono eseguite in parti, con una crescita incrementale delle funzionalità software. La dimensione di questi incrementi di funzionalità varia, alcuni prevedono pezzi più grandi e alcuni pezzi più piccoli. Gli incrementi di funzionalità possono essere anche molto piccoli come una singola modifica a una schermata dell'interfaccia utente o una nuova opzione di query.

nel modello di sviluppo iterativo gruppi di funzionalità sono specificati, progettati, sviluppati e testati insieme, in una serie di cicli spesso di durata fissa. Le iterazioni possono comportare modifiche alle funzionalità sviluppate nelle iterazioni precedenti, in linea con le modifiche nell’ambito del progetto.

Ogni iterazione rilascia software funzionante, che è un sottoinsieme crescente dell’insieme complessivo delle funzionalità, fino al rilascio del software finale o all’interruzione dello sviluppo.

Esempi includono:

- Rational Unified Process: ogni iterazione tende ad essere relativamente lunga (ad esempio da due a tre mesi) e di conseguenza gli incrementi delle funzionalità sono grandi, come ad esempio due o tre gruppi di funzionalità correlate.
- Scrum: ogni iterazione tende ad essere relativamente breve (ad esempio ore, giorni o poche settimane) e di conseguenza gli incrementi di funzionalità sono piccoli, come ad esempio alcuni miglioramenti e/o due o tre nuove funzionalità.
- Kanban: implementato con o senza iterazioni di durata fissa, che possono rilasciare sia una singola funzionalità completa sia un suo miglioramento, oppure possono raggruppare un insieme di funzionalità in un unico rilascio.
- Spirale: comporta la creazione di incrementi sperimentali, alcuni dei quali possono essere pesantemente rielaborati o addirittura abbandonati nelle attività di sviluppo successive.

Componenti o sistemi sviluppati utilizzando questi modelli spesso prevedono livelli di test che si sovrappongono e si iterano durante lo sviluppo. Idealmente, il testing di ogni funzionalità viene svolto applicando diversi livelli di test fino al relativo rilascio. In alcuni casi i team utilizzano il continuous delivery o il continuous deployment, che richiedono entrambi una significativa automazione di livelli multipli di test, come parte di pipeline di rilascio. Molti gruppi di sviluppo che utilizzano questi metodi includono anche il concetto di team auto-organizzati (self-organizing team), i quali possono modificare le modalità con cui le attività di test sono organizzate, così come le relazioni tra tester e sviluppatori.

Questi metodi sviluppano un sistema che cresce progressivamente e può essere rilasciato agli utenti finali per singola funzionalità, per singola iterazione, o in modo tradizionale come major-release. Ad ogni rilascio di un incremento software agli utenti finali, il regression testing aumenta di dimensioni in modo rilevante, man mano che il sistema cresce.

A differenza dei modelli sequenziali, i modelli iterativi e incrementali possono rilasciare software funzionante in poche settimane o anche in pochi giorni, ma possono anche rendere disponibile il prodotto completo di tutti i requisiti in un periodo di mesi o anche di anni.

Per ulteriori informazioni sul testing del software nel contesto dello sviluppo Agile, si veda il Syllabus ISTQB-CTFL-AT, Black 2017, Crispin 2008 e Gregory 2015.

2.1.2 Modelli del Ciclo di Vita dello Sviluppo Software nel Contesto

I modelli del ciclo di vita dello sviluppo software devono essere selezionati e adattati al contesto del progetto e alle caratteristiche del prodotto. Un modello appropriato di ciclo di vita dello sviluppo software dovrebbe essere selezionato e adattato in base all'obiettivo del progetto, al tipo di prodotto da sviluppare, alle priorità di business (ad esempio time-to-market), e ai rischi di prodotto e di progetto identificati. Ad esempio, lo sviluppo e testing di un piccolo sistema amministrativo interno dovrebbe differire dallo sviluppo e testing di un sistema safety-critical, come il sistema di controllo di frenata di un'automobile. Come altro esempio, problemi organizzativi e culturali possono in alcuni casi inibire la comunicazione tra i membri del team, impedendo lo sviluppo iterativo.

In base al contesto del progetto, può essere necessario combinare o riorganizzare i livelli e/o le attività di test. Ad esempio, per l'integrazione di un prodotto software Commercial Off-The-Shelf (COTS) in un sistema più grande, l'acquirente può eseguire il testing di interoperabilità a livello di test di integrazione dei sistemi (ad esempio integrazione con l'infrastruttura e altri sistemi) e a livello di test di accettazione (eseguendo testing funzionale e non-funzionale, User Acceptance Test e Operational Acceptance Test). Si veda il paragrafo 2.2 per dettagli sui livelli di test e il paragrafo 2.3 per dettagli sui tipi di test.

In aggiunta, i modelli del ciclo di vita dello sviluppo software possono essere combinati tra loro. Ad esempio, un V-Model può essere utilizzato per lo sviluppo e il testing dei sistemi di back-end e delle loro integrazioni, mentre un modello di sviluppo Agile può essere utilizzato per lo sviluppo e il testing dell'interfaccia utente (User Interface, UI) di front-end e delle funzionalità. Un modello di sviluppo a

spirale può essere utilizzato all'inizio di un progetto, applicando successivamente un modello di sviluppo incrementale, una volta completata la fase sperimentale.

Sistemi Internet of Things (IoT), che consistono di molti oggetti differenti, come dispositivi, prodotti e servizi, applicano in genere modelli del ciclo di vita dello sviluppo software separati per ciascun oggetto che lo compone. Questo comporta una sfida particolare nello sviluppo delle versioni del sistema Internet of Things. Inoltre, il ciclo di vita dello sviluppo software di tali oggetti pone maggiore enfasi sulle fasi successive del ciclo di vita dello sviluppo software, dopo l’avvio al loro utilizzo (ad esempio fasi di utilizzo operativo, di aggiornamento e di disinstallazione).

Le ragioni per cui i modelli di sviluppo software devono essere adattati al contesto del progetto e alle caratteristiche del prodotto possono essere:

- Differenza nei rischi di prodotto dei sistemi (che dipendono se un progetto è semplice o complesso)
- Molte unità di business possono essere parti di un progetto o di un programma (che richiede l’utilizzo di una combinazione dello sviluppo sequenziale e Agile)
- Tempi brevi per rilasciare sul mercato un prodotto (che richiede di unire diversi livelli di test e/o integrare tipi di test in livelli di test)

2.2 Livelli di Test

I livelli di test sono gruppi di attività di test che sono organizzate e gestite insieme. Ogni livello di test è un’istanza del processo di test, che consiste delle attività descritte nel paragrafo 1.4, svolte in relazione al software per un determinato livello di sviluppo (da singole unità o componenti a sistemi completi o, se applicabile, a sistemi di sistemi). I livelli di test sono correlati ad altre attività all'interno del ciclo di vita dello sviluppo software.

I livelli di test utilizzati in questo Syllabus sono:

- Testing di componente
- Testing di integrazione
- Testing di sistema
- Testing di accettazione

I livelli di test sono caratterizzati dai seguenti attributi:

- Obiettivi specifici
- Base di test, usata come riferimento per derivare i test case
- Oggetto di test (cioè cosa è sotto test)
- Difetti e failure tipici
- Approcci specifici e responsabilità

Per ogni livello di test è richiesto un ambiente di test adeguato. Nel testing di accettazione, ad esempio, un’ambiente di test simile alla produzione è l'ideale, mentre nel testing di componente gli sviluppatori usano normalmente il proprio ambiente di sviluppo.

2.2.1 Testing di Componente

Obiettivi del Testing di Componente

Il testing di componente (chiamato anche test di unità o di modulo) si focalizza su componenti che sono testabili separatamente. Gli obiettivi del testing di componente includono:

- Ridurre il rischio

- Verificare se i comportamenti funzionali e non-funzionali del componente corrispondono a quanto progettato e specificato
- Aumentare la confidenza nella qualità del componente
- Rilevare difetti nel componente
- Prevenire difetti nei livelli di test più alti

In alcuni casi, specialmente nei modelli di sviluppo incrementali e iterativi (ad es. Agile), dove le modifiche al codice sono continue, i regression test di componente automatizzati svolgono un ruolo chiave nel fornire fiducia che le modifiche non abbiano introdotto difetti nei componenti esistenti.

Il testing di componente viene spesso eseguito separatamente dal resto del sistema, in base al sistema e al modello del ciclo di vita dello sviluppo software, e questo può richiedere oggetti fittizi (mock object), virtualizzazione dei servizi, test harness, stub e driver. Il testing di componente può coprire le caratteristiche funzionali (ad es. la correttezza dei calcoli), le caratteristiche non-funzionali (ad es. ricerca di memory leak) e le proprietà strutturali (ad es. testing delle decisioni).

Base di Test

Esempi di prodotti di lavoro che possono essere utilizzati come base di test per il testing di componente includono:

- Progettazione di dettaglio
- Codice
- Modello dati
- Specifiche di componente

Oggetti di Test

Oggetti di test tipici del testing di componente includono:

- Componenti, unità o moduli
- Codice e strutture dati
- Classi
- Moduli di database

Difetti e failure tipici

Esempi di difetti e failure tipici del testing di componente includono:

- Funzionalità non corrette (ad es. implementazione differente da quanto descritto nelle specifiche di progettazione)
- Problemi di flusso dati (data flow)
- Codice o logica non corretti

I difetti vengono generalmente corretti non appena vengono trovati, spesso senza una gestione formale di Defect Management. Tuttavia, quando gli sviluppatori tracciano i difetti, è possibile fornire informazioni importanti per la root cause analysis e il miglioramento del processo.

Approcci specifici e responsabilità

Il testing di componente viene solitamente eseguito dallo sviluppatore che ha scritto il codice. Richiede almeno l’accesso al codice dell’oggetto di test. Gli sviluppatori possono alternare l’attività di sviluppo del componente con l’attività di ricerca e correzione dei difetti. Gli sviluppatori spesso scrivono ed eseguono i test dopo aver scritto il codice per un componente. Tuttavia, in particolare nello sviluppo Agile, la scrittura di test case di componente automatizzati può precedere la scrittura del codice dell’applicativo.

Ad esempio lo sviluppo Test-Driven Development (TDD) è fortemente iterativo ed è basato su cicli di sviluppo di test case automatizzati, e successiva implementazione e integrazione di piccoli pezzi di codice, e quindi esecuzione dei test di componente, correzione di eventuali problemi e refactoring del codice. Questo processo continua fino a quando il componente è stato completamente sviluppato e tutti i test di componente sono eseguiti con successo. Il Test-Driven Development è un esempio di approccio Test-First. Benché il Test-Driven Development abbia avuto origine dall’approccio Agile eXtreme Programming (XP), si è esteso in altri approcci Agile e anche nei modelli di ciclo di vita sequenziali (si veda il Syllabus ISTQB-CTFL-AT).

2.2.2 Testing di Integrazione

Obiettivi del Testing di Integrazione

Il testing di integrazione si concentra sulle interazioni tra componenti o sistemi. Gli obiettivi del testing di integrazione includono:

- Ridurre il rischio
- Verificare se i comportamenti funzionali e non-funzionali delle interfacce corrispondono a quanto progettato e specificato
- Aumentare la confidenza nella qualità delle interfacce
- Rilevare difetti (nelle interfacce stesse o all'interno dei componenti o dei sistemi)
- Prevenire i difetti nei livelli di test più alti

Come per il testing di componente, in alcuni casi i regression test di integrazione automatizzati assicurano che le modifiche non abbiano introdotto difetti nelle interfacce, nei componenti o nei sistemi esistenti.

Esistono due diversi livelli di test di integrazione descritti in questo Syllabus, che possono essere condotti su oggetti di test di varie dimensioni:

- Il testing di integrazione dei componenti si focalizza su interazioni e interfacce tra componenti integrati. Il testing di integrazione dei componenti viene eseguito dopo il testing di componente ed è generalmente automatizzato. Nello sviluppo iterativo e incrementale i test di integrazione dei componenti sono di solito parte del processo di continuous integration.
- Il testing di integrazione dei sistemi si focalizza su interazioni e interfacce tra sistemi, package e microservizi. Il testing di integrazione dei sistemi può anche coprire le interazioni e le interfacce con organizzazioni esterne (ad es. servizi web). In questo caso l'organizzazione che è fornitore non controlla le interfacce esterne e questo può creare diverse sfide per il testing (ad es. garantendo che i difetti bloccanti per il test nel codice dell'organizzazione esterna siano risolti, installando gli ambienti di test, ecc.). Il testing di integrazione dei sistemi può essere svolto dopo il testing di sistema o in parallelo alle attività del testing di sistema in corso (sia nello sviluppo sequenziale che nello sviluppo iterativo e incrementale).

Base di Test

Esempi di prodotti di lavoro che possono essere utilizzati come base di test per il testing di integrazione includono:

- Progettazione software e di sistema
- Sequence diagram
- Specifiche di interfacce e protocolli di comunicazione
- Use case (caso d’uso)
- Architettura a livello di componente o sistema
- Workflow
- Definizioni delle interfacce esterne

Oggetti di Test

Oggetti di test tipici del testing di integrazione includono:

- Sottosistemi
- Database
- Infrastruttura
- Interfacce
- API
- Microservizi

Difetti e failure tipici

Esempi di difetti e failure tipici del testing di integrazione dei componenti includono:

- Dati errati, dati mancanti o codifica errata dei dati
- Sequenza o sincronizzazione errata delle chiamate di interfaccia
- Disallineamenti di interfaccia
- Failure nella comunicazione tra componenti
- Failure di comunicazione tra componenti non gestita o gestita in modo non appropriato
- Assunzioni errate sul significato, sulle unità di misure o sui valori limite dei dati trasferiti tra componenti

Esempi di difetti e failure tipici del testing di integrazione dei sistemi includono:

- Strutture inconsistenti dei messaggi scambiati tra sistemi
- Dati errati, dati mancanti o codifica errata dei dati
- Disallineamenti di interfaccia
- Failure nella comunicazione tra sistemi
- Failure di comunicazione tra sistemi non gestita o gestita in modo non appropriato
- Assunzioni errate sul significato, sulle unità di misura o sui valori limite dei dati trasferiti tra sistemi
- Mancato rispetto delle normative obbligatorie di sicurezza

Approcci specifici e responsabilità

I test di integrazione dei componenti e i test di integrazione dei sistemi dovrebbero concentrarsi sull'integrazione stessa. Ad esempio, se si integra il modulo A con il modulo B, i test dovrebbero focalizzarsi sulla comunicazione tra i moduli, non sulle funzionalità dei singoli moduli, che dovrebbero essere state coperte durante il testing di componente. Se il sistema X viene integrato con il sistema Y, i test dovrebbero focalizzarsi sulla comunicazione tra i sistemi, non sulle funzionalità dei singoli sistemi, che dovrebbero essere state coperte durante il testing di sistema. Sono applicabili tipi di test funzionali, non-funzionali e strutturali.

Il testing di integrazione dei componenti è spesso sotto la responsabilità degli sviluppatori. Il testing di integrazione dei sistemi è generalmente sotto la responsabilità dei tester. Idealmente, i tester che eseguono il testing di integrazione dei sistemi dovrebbero conoscere l'architettura dei sistemi e dovrebbero aver collaborato alla pianificazione dell'integrazione.

Se i test di integrazione e la strategia di integrazione sono pianificati prima dello sviluppo dei componenti o dei sistemi, questi componenti o sistemi possono essere sviluppati nell'ordine richiesto per condurre un testing più efficiente. Le strategie sistematiche di integrazione possono essere basate sull'architettura dei sistemi (ad es. top-down e bottom-up), sulle attività funzionali, sulle sequenze di

elaborazione delle transazioni o su qualche altro aspetto dei sistemi o dei componenti. Per semplificare l'isolamento dei difetti e per rilevare i difetti in anticipo, l'integrazione dovrebbe normalmente essere incrementale (prevedendo ad ogni passo l’inserimento di un numero ridotto di componenti o sistemi) piuttosto che "big bang" (dove l'integrazione di tutti componenti o sistemi avviene in un unico passo). Un'analisi del rischio delle interfacce più complesse può aiutare a focalizzare il testing di integrazione.

Maggiore è l’ambito dell'integrazione, maggiore è la difficoltà di isolare i difetti relativi a uno specifico componente o sistema, e questo può causare un aumento del rischio e un tempo aggiuntivo per la risoluzione dei problemi (trubleshooting). Questo è uno dei motivi per cui la Continuous Integration, in cui il software viene integrato componente per componente (ovvero per integrazione funzionale), è diventata una pratica comune. Tale continuous integration spesso include regression test automatizzati, applicati idealmente a livelli multipli di test.

2.2.3 Testing di Sistema

Obiettivi del testing di sistema

Il testing di sistema si concentra sul comportamento e sulle capacità di un intero sistema o prodotto, spesso considerando le attività end-to-end che il sistema può eseguire e i comportamenti non-funzionali che sono visibili durante l'esecuzione di tali attività. Gli obiettivi del testing di sistema includono:

- Ridurre il rischio
- Verificare se i comportamenti funzionali e non-funzionali del sistema corrispondono a quanto progettato e specificato
- Validare che il sistema sia completo e funzionerà come previsto
- Aumentare la confidenza nella qualità del sistema completo
- Rilevare difetti
- Prevenire difetti nei livelli di test più alti o in produzione

Per alcuni sistemi, anche la verifica della qualità dei dati può essere un obiettivo. Come nel testing di componente e di integrazione, in alcuni casi i regression test automatizzati di sistema forniscono la confidenza che le modifiche non abbiano alterato le funzionalità esistenti o le funzionalità end-to-end. Il testing di sistema spesso produce informazioni che vengono utilizzate dagli stakeholder per prendere decisioni sul rilascio. Il testing di sistema può anche dover soddisfare requisiti legali, requisiti normativi o standard.

L'ambiente di test dovrebbe idealmente corrispondere all’ambiente target finale o all'ambiente di produzione.

Base di Test

Esempi di prodotti di lavoro che possono essere utilizzati come base di test per il testing di sistema includono:

- Specifiche dei requisiti software e di sistema (funzionali e non-funzionali)
- Report di analisi del rischio
- Use case
- Epic e user story
- Modelli di comportamento del sistema
- State diagram
- Manuali utente e di sistema

Oggetti di Test

Oggetti di test tipici del testing di sistema includono:

- Applicativi
- Sistemi hardware/software
- Sistemi operativi
- System Under Test (SUT)
- Configurazione di sistema e dati di configurazione

Difetti e failure tipici

Esempi di difetti e failure tipici del testing di sistema includono:

- Calcoli errati
- Comportamento funzionale o non-funzionale del sistema errato o non atteso
- Data flow (flussi dati) e/o control flow (flussi di controllo) errati all'interno del sistema
- Failure sull’esecuzione corretta e completa delle caratteristiche funzionali end-to-end
- Failure sul funzionamento corretto del sistema negli ambienti di test
- Failure sul funzionamento del sistema rispetto a quanto descritto nei manuali utente e di sistema

Approcci specifici e responsabilità

Il testing di sistema dovrebbe concentrarsi sul comportamento complessivo end-to-end del sistema nel suo insieme, sia dal punto di vista funzionale che non-funzionale. Il testing di sistema dovrebbe utilizzare le tecniche più appropriate (si veda il capitolo 4) in base agli aspetti del sistema sotto test. Ad esempio, è possibile creare una tabella delle decisioni per verificare se il comportamento funzionale sia quello descritto nelle regole di business.

Il testing di sistema è normalmente svolto da tester indipendenti facendo grande affidamento alle specifiche. Difetti nelle specifiche (ad es. user story mancanti, requisiti di business formulati in modo errato ecc.) possono causare mancanza di comprensione o disaccordo sul comportamento atteso del sistema. Tali situazioni possono generare falsi positivi e falsi negativi che, rispettivamente, fanno perdere tempo e riducono l'efficacia nel rilevamento dei difetti. Il coinvolgimento anticipato dei tester nel raffinamento delle user story, o le attività di testing statico, cosi come le review, aiutano a ridurre l'incidenza di tale situazioni.

2.2.4 Testing di Accettazione

Obiettivi del testing di accettazione

Il testing di accettazione, come il testing di sistema, si focalizza normalmente sul comportamento e sulle capacità di un sistema o prodotto completo. Gli obiettivi del testing di accettazione includono:

- Stabilire confidenza sulla qualità del sistema completo
- Validare che il sistema sia completo e funzioni come previsto
- Verificare che i comportamenti funzionali e non-funzionali del sistema siano quelli specificati

Il testing di accettazione può produrre informazioni per valutare la readiness del sistema al rilascio e all’utilizzo da parte del cliente (utente finale). Durante il testing di accettazione possono essere rilevati difetti, benché ciò non sia spesso un obiettivo, dato che trovare un numero significativo di difetti durante il testing di accettazione può essere considerato in alcuni casi un importante rischio di progetto. Il testing di accettazione può anche dover soddisfare requisiti legali, requisiti normativi o standard.

Tipi comuni di testing di accettazione includono:

- User Acceptance Test (UAT)
- Operational Acceptance Test (OAT)

- Testing di accettazione contrattuale e normativo
- Alpha test e Beta test

Ognuno di questi è descritto nei seguenti quattro sottoparagrafi.

User Acceptance Test (UAT)

Gli User Acceptance Test del sistema è normalmente focalizzato sulla validazione dell'idoneità all'utilizzo (fitness for use) del sistema da parte degli utenti previsti, in un ambiente operativo reale o simulato. L'obiettivo principale è prendere confidenza e valutare che gli utenti possano utilizzare il sistema per soddisfare le loro esigenze e i requisiti, ed eseguire i processi di business con il minino di difficoltà, costi e rischi.

Operational Acceptance Test (OAT)

Gli Operational Acceptance Test del sistema da parte dello staff di esercizio (operations) o di system administration vengono normalmente svolti in un ambiente di produzione (simulato). I test si focalizzano sugli aspetti operativi e possono includere:

- Testing del backup e ripristino
- Installazione, disinstallazione e aggiornamento
- Disaster recovery
- Gestione utenti
- Attività di manutenzione
- Attività di caricamento e migrazione dei dati
- Verifica delle vulnerabilità di sicurezza
- Performance test

L'obiettivo principale degli Operational Acceptance Test è di aumentare la fiducia che gli operatori o i system administrator possano mantenere il sistema funzionante in modo appropriato per gli utenti nell'ambiente operativo, anche in condizioni difficili o nelle eccezioni.

Testing di accettazione contrattuale e normativo

Il testing di accettazione contrattuale viene svolto rispetto a criteri di accettazione definiti nel contratto per produrre software customizzato. I criteri di accettazione dovrebbero essere definiti quando le parti finalizzano il contratto. Il testing di accettazione contrattuale viene spesso eseguito dagli utenti o da tester indipendenti.

Il testing di accettazione normativo viene svolto rispetto alle normative a cui il sistema deve aderire, come normative governative, legali o di safety. Il testing di accettazione normativo è spesso svolto da utenti o tester indipendenti, e qualche volta i risultati sono validati attraverso un audit o certificati da specifiche agenzie regolatrici.

L'obiettivo principale del testing di accettazione contrattuale e normativo è di aumentare la fiducia che sia stata raggiunta la conformità contrattuale o normativa.

Alpha test e Beta test

Gli Alpha test e Beta test sono generalmente utilizzati dalle aziende produttrici di software Commercial Off-The-Shelf (COTS), quando vogliono ricevere feedback da utenti potenziali o esistenti, da clienti e/o da operatori prima che il prodotto software venga immesso sul mercato. L’Alpha test viene svolto presso la sede del produttore, non dal team di sviluppo ma da clienti potenziali o esistenti, e/o da operatori, oppure da un team di test indipendente. Il Beta test viene eseguito, da clienti potenziali o esistenti, e/o da operatori, presso le proprie sedi. Il Beta test può essere condotto dopo l’Alpha test oppure può svolgersi senza aver svolto l’Alpha test in precedenza.

Un obiettivo degli Alpha e Beta test è aumentare la fiducia, da parte degli operatori e/o clienti potenziali o esistenti, di poter utilizzare il sistema in condizioni normali, quotidiane e negli ambienti operativi, in

modo da raggiungere i loro obiettivi con il minimo delle difficoltà, costi e rischi. Un altro obiettivo può essere il rilevamento di difetti relativi alle condizioni e all'ambiente in cui verrà utilizzato il sistema, specialmente quando tali condizioni e ambienti siano difficili da replicare da parte del team di sviluppo.

Base di Test

Esempi di prodotti di lavoro che possono essere utilizzati come base di test per il testing di accettazione includono:

- Processi di business
- Requisiti utente o di business
- Normative, contratti legali e standard
- Use case e/o user story
- Requisiti di sistema
- Documentazione di sistema o degli utenti
- Procedure di installazione
- Report di analisi del rischio

Inoltre, come base di test per derivare i test case relativi agli Operational Acceptance Test, possono essere utilizzati uno o più dei seguenti prodotti di lavoro:

- Procedure di backup e restore
- Procedure di disaster recovery
- Requisiti non-funzionali
- Documentazione operativa
- Istruzioni per la distribuzione e installazione
- Obiettivi prestazionali
- Package di database
- Standard o normative di sicurezza

Oggetti di Test

Oggetti di test tipici per qualsiasi tipo di testing di accettazione includono:

- System Under Test (SUT)
- Configurazione del sistema e dati di configurazione
- Processi di business per un sistema completamente integrato
- Sistemi di recovery e hot site (per il testing della busineess continuity e del disaster recovery)
- Processi operativi e di manutenzione
- Form
- Report
- Dati di produzione esistenti e convertiti

Difetti e failure tipici

Esempi di difetti e failure tipici per qualsiasi tipo di testing di accettazione includono:

- I workflow di sistema non soddisfano i requisiti di business o dell'utente
- Le regole di business non sono implementate correttamente

- Il sistema non soddisfa i requisiti contrattuali o normativi
- Failure non-funzionali, quali vulnerabilità di sicurezza, inadeguata efficienza delle prestazioni sotto carichi elevati, oppure operazioni non appropriate su una delle piattaforme supportate

Approcci specifici e responsabilità

Il testing di accettazione è spesso sotto la responsabilità dei clienti, utenti di business, Product Owner oppure operatori di un sistema, ma possono essere coinvolti anche altri stakeholder.

Il testing di accettazione è spesso previsto come ultimo livello di test in un ciclo di vita di sviluppo sequenziale, ma può svolgersi anche in altri momenti, ad esempio:

- Il testing di accettazione di un prodotto software COTS può essere eseguito dopo l’installazione o l’ integrazione
- Il testing di accettazione di un nuovo miglioramento funzionale può svolgersi prima del testing di sistema

Nello sviluppo iterativo, i team di progetto possono applicare varie forme di testing di accettazione durante e alla fine di ogni iterazione, ad esempio focalizzate sulla verifica di una nuova funzionalità rispetto ai propri criteri di accettazione, o focalizzati sulla validazione di una nuova funzionalità rispetto alle esigenze degli utenti. Inoltre, gli Alpha e Beta test possono svolgersi al termine di ogni iterazione, dopo il completamento di ogni iterazione o dopo una serie di iterazioni. Inoltre, tutti i tipi di testing di accettazione (User Acceptance Test, Operational Acceptance Test, testing di accettazione normativa e contrattuale, Alpha e Beta Test) possono essere eseguiti alla chiusura di ogni iterazione, dopo il completamento di ogni iterazione o dopo una serie di iterazioni.

2.3 Tipi di Test

Un tipo di test è un gruppo di attività di test con lo scopo di testare caratteristiche specifiche di un sistema software, o di una parte di un sistema, sulla base di specifici obiettivi di test. Tali obiettivi possono includere:

- Valutare le caratteristiche di qualità funzionali, come completezza, correttezza e appropriatezza
- Valutare le caratteristiche di qualità non-funzionali, come affidabilità, efficienza delle prestazioni, sicurezza, compatibilità e usabilità
- Valutare se la struttura o l'architettura del componente o sistema è corretta, completa e conforme a quanto specificato
- Valutare gli effetti delle modifiche, ad esempio confermando che i difetti sono stati corretti (testing confermativo) e verificando che non siano state introdotte modifiche non intenzionali al comportamento, a seguito di modifiche software o dell’ambiente (regression testing)

2.3.1 Testing Funzionale

Il testing funzionale di un sistema include i test che valutano le funzioni che il sistema dovrebbe svolgere. I requisiti funzionali possono essere descritti in prodotti di lavoro come le specifiche dei requisiti di business, epic, user story, use case o specifiche funzionali, oppure potrebbero non essere documentati. Le funzioni rappresentano "cosa" il sistema dovrebbe fare.

I test funzionali dovrebbero essere eseguiti a tutti i livelli di test (ad es. i test per i componenti possono basarsi su una specifica dei componenti), sebbene il focus sia diverso per ogni livello di test (si veda il paragrafo 2.2).

Il testing funzionale considera il comportamento del software, quindi è possibile utilizzare tecniche black-box per derivare condizioni di test e test case relativi alla funzionalità del componente o del sistema (si veda il paragrafo 4.2).

La precisione del testing funzionale può essere misurata attraverso la copertura funzionale. La copertura funzionale è la misura con cui una funzionalità è stata esercitata dai test ed è espressa come percentuale del tipo (o tipi) di elementi coperti dal testing. Ad esempio, utilizzando la tracciabilità tra test e requisiti funzionali, può essere calcolata la percentuale di requisiti che sono stati esercitati dai test, identificando potenzialmente lacune nella copertura.

La progettazione e l'esecuzione dei test funzionali possono coinvolgere figure con skill o conoscenze speciali, come la conoscenza del particolare problema di business risolto dal software (ad es. software di modellazione geologica per aziende che estraggono petrolio e gas)

2.3.2 Testing Non-Funzionale

Il testing non-funzionale di un sistema valuta le caratteristiche di software e sistemi, come usabilità, efficienza delle prestazioni o sicurezza. Si faccia riferimento allo standard ISO (ISO/IEC 25010) per una classificazione delle caratteristiche di qualità di un prodotto software. Il testing non-funzionale è il testing di "quanto bene" si comporta il sistema.

Contrariamente a comuni percezioni errate, il testing non-funzionale può, e spesso dovrebbe, essere svolto a tutti i livelli di test e il prima possibile. La scoperta tardiva di difetti non funzionali può essere estremamente pericolosa per il successo di un progetto.

Le tecniche black-box (si veda il paragrafo 4.2) possono essere utilizzate per derivare condizioni di test e test case per il testing non-funzionale. Ad esempio, l'analisi dei valore limite può essere utilizzata per definire le condizioni di stress per i performance test.

La completezza del testing non-funzionale può essere misurata attraverso la copertura non-funzionale. La copertura non-funzionale è la misura con cui un certo tipo di elemento non-funzionale è stato esercitato dai test ed è espressa come percentuale del tipo (o tipi) di elementi coperti dal testing. Ad esempio, utilizzando la tracciabilità tra test e dispositivi supportati per un'applicazione mobile, può essere calcolata la percentuale di dispositivi mobile che sono stati esercitati dai test di compatibilità, identificando potenzialmente lacune nella copertura.

La progettazione e l'esecuzione dei test non-funzionali possono coinvolgere figure con skill o conoscenze speciali, come la conoscenza delle debolezze intrinseche di una progettazione o di una tecnologia (ad es. vulnerabilità di sicurezza associate a particolari linguaggi di programmazione) o di una particolare user base (ad es. i profili di utenti dei sistemi di gestione delle strutture sanitarie).

Si faccia riferimento ai Syllabi ISTQB-CTAL-TA, ISTQB-CTAL-TTA, ISTQB-CTAL-SEC e ad altri moduli specialist di ISTQB® per maggiori dettagli riguardanti il testing delle caratteristiche di qualità non-funzionali.

2.3.3 Testing White-box

Il testing white-box deriva i test case in base alla struttura interna o all'implementazione del sistema. La struttura interna può includere codice, architettura, workflow e/o data flow all'interno del sistema (si veda il paragrafo 4.3).

La completezza del testing white-box può essere misurata attraverso la copertura strutturale. La copertura strutturale è la misura di quanto un tipo di elemento strutturale è stato esercitato dai test, ed è espressa come percentuale del tipo di elemento coperto dal testing.

A livello di test di componente, la copertura del codice è basata sulla percentuale di codice del componente che è stato testato e può essere misurata in base a diversi aspetti del codice (elementi di copertura), come la percentuale di istruzioni eseguibili o la percentuale di risultati decisionali testati nel componente. Questi tipi di copertura sono chiamati in modo generico copertura del codice. A livello di test di integrazione dei componenti, il testing white-box può basarsi sull'architettura del sistema, come ad esempio le interfacce tra i componenti, e la copertura strutturale può essere misurata in termini di percentuale di interfacce esercitate dai test.

La progettazione ed esecuzione det test white-box possono richiedere skill o conoscenze speciali, come il modo con cui il codice viene sviluppato, come vengono memorizzati i dati (ad es. per valutare

possibili query di database), come utilizzare gli strumenti di copertura, e come interpretare correttamente i risultati.

2.3.4 Testing Relativo a Modifiche

Quando vengono apportate modifiche a un sistema, sia per correggere un difetto sia per modificare o aggiungere nuove funzionalità, dovrebbe essere effettuato del testing per confermare che le modifiche abbiano corretto il difetto o implementato correttamente la funzionalità, e non abbiano causato conseguenze impreviste (side-effect) non desiderate.

- **Testing confermativo**: dopo aver corretto un difetto, il software può essere testato con tutti i test case falliti a causa del difetto, che dovrebbero essere rieseguiti sulla nuova versione software. Il software può anche essere testato con nuovi test, per coprire le modifiche necessarie per correggere il difetto. Almeno i passi per riprodurre i failure causati dal difetto devono essere rieseguiti sulla nuova versione software. Lo scopo di un test confermativo è di confermare che il difetto originale è stato corretto con successo.
- **Regression Testing**: è possibile che una modifica implementata in una parte del codice, sia essa una correzione o un altro tipo di modifica, possa involontariamente influenzare il comportamento di altre parti del codice, nello stesso componente, in altri componenti dello stesso sistema o anche in altri sistemi. Le modifiche possono includere modifiche all'ambiente, come una nuova versione di un sistema operativo o di un sistema di database management. Tali side-effect indesiderati sono chiamati regressioni. Il regression testing prevede l'esecuzione di test per rilevare gli eventuali side-effect indesiderati.

Il testing confermativo e il regression testing sono svolti a tutti i livelli di test.

Soprattutto nei cicli di sviluppo iterativo e incrementale (ad es. Agile), le nuove funzionalità, le modifiche a funzionalità esistenti e il refactoring del codice comportano frequenti modifiche al codice, richiedendo quindi anche il testing relativo a tali modifiche. A causa della natura evolutiva del sistema, i test confermativi e i regression test sono molto importanti. Questo è particolarmente rilevante per i sistemi Internet of Things (IoT) in cui singoli oggetti (ad es. dispositivi) vengono frequentemente aggiornati o sostituiti.

Le test suite di regression test vengono eseguite molte volte e in genere evolvono lentamente, pertanto il regression testing è un candidato molto importante per l'automazione. L'automazione di questi test dovrebbe iniziare presto nel progetto (si veda il capitolo 6).

2.3.5 Tipi di Test e Livelli di Test

È possibile eseguire uno qualsiasi dei tipi di test sopra menzionati a qualsiasi livello di test. Per meglio illustrare questo, sono forniti di seguito esempi di test funzionali, test non-funzionali, test white-box e test relativi a modifiche, per tutti i livelli di test, con riferimento ad una applicazione bancaria, a partire dai test funzionali:

- Per il testing di componente, i test sono progettati tenendo conto di come un componente dovrebbe calcolare l’interesse composto.
- Per il testing di integrazione dei componenti, i test sono progettati tenendo conto di come le informazioni del titolare del conto inserite nell'interfaccia utente vengono passate alla logica di business.
- Per il testing di sistema, i test sono progettati tenendo conto di come i titolari dei conti possono richiedere una linea di credito sui loro conti correnti.
- Per il testing di integrazione dei sistemi, i test sono progettati tenendo conto di come il sistema utilizza un microservice esterno per verificare un credito del titolare di un conto.
- Per il testing di accettazione, i test sono progettati tenendo conto di come il bancario (Interfaccia utente) gestisce l'accettazione o il rifiuto di una richiesta di credito.

Di seguito sono riportati alcuni esempi di test non-funzionali:

- Per il testing di componente, i performance test sono progettati per valutare il numero di cicli di CPU richiesti per eseguire un calcolo complesso dell'interesse totale.
- Per il testing di integrazione dei componenti, i test di sicurezza sono progettati per le vulnerabilità di buffer overflow, a causa dei dati passati dall'interfaccia utente alla logica di business.
- Per il testing di sistema, i test di portabilità sono progettati per verificare se il presentation layer lavora su tutti i browser e dispositivi mobili supportati.
- Per il testing di integrazione dei sistemi, i test di affidabilità sono progettati per valutare la robustezza del sistema, quando il microservice relativo al credito risponda in modo errato.
- Per il testing di accettazione, i test di usabilità sono progettati per valutare l'accessibilità dell’interfaccia utente (bancario) di elaborazione del credito alle persone con disabilità.

I seguenti sono esempi di test white-box:

- Per il testing di componente, i test sono progettati per ottenere una copertura completa delle istruzioni e delle decisioni (si veda il paragrafo 4.3), per tutti i componenti che eseguono calcoli finanziari.
- Per il testing di integrazione dei componenti, i test sono progettati per esercitare i modi in cui ciascuna schermata dell'interfaccia del browser passa i dati alla schermata successiva e alla logica di business.
- Per il testing di sistema, i test sono progettati per coprire le sequenze di pagine Web che possono presentarsi durante un’operazione della linea di credito.
- Per il testing di integrazione dei sistemi, i test sono progettati per esercitare tutte le possibili richieste inviate al microservice di valutazione del credito.
- Per il testing di accettazione, i test sono progettati per coprire tutte le strutture dei file di dati finanziari e i range di valori per i trasferimenti bank-to-bank.

Infine, i seguenti sono esempi di test relativi a modifiche:

- Per il testing di componente, vengono creati regression test automatizzati per ciascun componente per essere inclusi nel framework di continuous integration .
- Per il testing di integrazione dei componenti, i test sono progettati per confermare le correzioni di difetti relativi all'interfaccia, non appena tali correzioni vengono caricate nel repository del codice.
- Per il testing di sistema, tutti i test per un determinato workflow vengono rieseguiti, se qualsiasi schermata di quel workflow è stata modificata.
- Per il testing di integrazione dei sistemi, i test dell'applicativo che interagisce con il microservice di valutazione del credito vengono rieseguiti quotidianamente, come parte del continuous deployment di quel microservice.
- Per il testing di accettazione, tutti i test precedentemente falliti sono rieseguiti, dopo che un difetto rilevato dai test di accettazione è stato corretto.

Benché questo paragrafo fornisca esempi di ogni tipo di test per tutti i livelli, non è necessario, per tutti i software, che venga svolto ogni tipo di test per tutti i livelli. Tuttavia, è importante eseguire i tipi di test applicabili a ciascun livello, in particolare a partire dal livello più basso a cui il tipo di test può essere applicato.

2.4 Testing di Manutenzione

Dopo essere stati rilasciati in ambiente di produzione, software e sistemi devono essere manutenuti. Modifiche di vario tipo sono quasi inevitabili nei software e nei sistemi rilasciati, per correggere difetti scoperti durante l’utilizzo operativo, per aggiungere nuove funzionalità, oppure per eliminare o

modificare le funzionalità già rilasciate. La manutenzione è necessaria anche per preservare o migliorare le caratteristiche di qualità non-funzionali del componente o del sistema nel corso della sua vita, in particolare l’efficienza delle prestazioni, la compatibilità, l'affidabilità, la sicurezza e la portabilità.

Quando durante la manutenzione viene apportata qualsiasi modifica, dovrebbero essere eseguiti i test di manutenzione, sia per valutare il successo delle modifiche implementate, sia per verificare eventuali side-effect (regressioni) in parti del sistema rimaste invariate (che sono di solito la maggior parte del sistema).La manutenzione può comportare rilasci pianificati e rilasci non pianificati (hot fix).

Un rilascio di manutenzione può richiedere test di manutenzione a livelli multipli di test, utilizzando diversi tipi di test, in base al suo ambito. L’ambito del testing di manutenzione dipende da:

- Il grado di rischio della modifica, ad esempio, il grado con cui il software modificato comunica con altri componenti o sistemi
- La dimensione del sistema esistente
- La dimensione della modifica

2.4.1 Trigger per la Manutenzione

Esistono diversi motivi per cui la manutenzione del software, e quindi il testing di manutenzione, debba essere effettuato, sia per modifiche pianificate che non pianificate.

Si possono classificare i trigger per la manutenzione come segue:

- **Modifica**, come ad esempio miglioramenti pianificati (ad es. rilasci periodici o release-based), modifiche correttive e di emergenza, cambiamenti dell'ambiente operativo (come aggiornamenti pianificati del sistema operativo o del database), aggiornamenti del software COTS e patch per difetti e vulnerabilità
- **Migrazione**, come ad esempio da una piattaforma all'altra, che può richiedere test operativi nel nuovo ambiente nonché del software modificato, o test di conversione dei dati quando sono migrati i dati da un'altra applicazione al sistema che viene manutenuto
- **Ritiro**, come ad esempio quando un’applicazione raggiunge la fine della sua vita.
    - Quando un'applicazione o un sistema viene ritirato, può essere necessario effettuare il test di migrazione dei dati o della loro archiviazione, se sono richiesti periodi di long-retention.
    - Potrebbe anche essere necessario il testing delle procedure di retrieve/restore dei dati per il periodo di long-retention
    - Può essere richiesto il regression testing per garantire il corretto funzionamento delle funzionalità ancora in servizio.

Per i sistemi Internet of Things (IoT), il testing di manutenzione può essere innescato dall'introduzione nel sistema globale di sistemi (Thing) completamente nuove oppure modificate, come dispositivi hardware e servizi software. Il testing di manutenzione per tali sistemi pone particolare enfasi sul testing di integrazione a differenti livelli (ad es. livello di rete, livello applicativo) e sugli aspetti di sicurezza, in particolare quelli relativi ai dati personali.

2.4.2 Analisi degli Impatti per la Manutenzione

L'analisi degli impatti (change impact analysis) valuta le modifiche apportate in un rilascio di manutenzione per individuare le conseguenze attese e i possibili side-effect della modifica, e per identificare le aree del sistema che saranno impattate dalle modifiche stesse. L'analisi degli impatti può aiutare anche a identificare l'impatto di una modifica sui test esistenti. È necessario testare per le regressioni sia i side-effect che le aree del sistema impattate, eventualmente dopo aver aggiornato gli eventuali test esistenti impattati dalla modifica.

L'analisi degli impatti può essere svolta prima che venga apportata una modifica, per aiutare a decidere se la modifica debba essere implementata, in base alle potenziali conseguenze in altre aree del sistema.

L'analisi degli impatti può essere difficile se:

- Le specifiche non sono aggiornate o sono mancanti (ad es. requisiti di business, user story, architettura)
- I test case non sono documentati o non sono aggiornati
- La tracciabilità bidirezionale tra test e base di test non è stata manutenuta
- Il supporto degli strumenti è debole o inesistente
- Le persone coinvolte non hanno conoscenze del dominio e/o del sistema
- È stata dedicata scarsa attenzione alla manutenibilità del software durante lo sviluppo

3. Testing Statico – 135 minuti

Parole Chiave

analisi statica, ispezione, lettura perspective-based, review (revisione), review ad hoc, review checklist-based, review role-based, review scenario-based, review formale, review informale, review tecnica, testing dinamico, testing statico, walkthrough

Obiettivi di Apprendimento per il Testing Statico

3.1 Fondamenti del Testing Statico

FL-3.1.1 (K1) Riconoscere i tipi di prodotti di lavoro del software che possono essere esaminati dalle diverse tecniche di testing statico

FL-3.1.2 (K2) Utilizzare esempi per descrivere il valore del testing statico

FL-3.1.3 (K2) Spiegare la differenza tra tecniche statiche e tecniche dinamiche, considerando obiettivi, tipi di difetti da identificare e ruolo di queste tecniche all'interno del ciclo di vita del software

3.2 Processo di Review (Revisione)

FL-3.2.1 (K2) Riassumere le attività del processo di review di un prodotto di lavoro

FL-3.2.2 (K1) Riconoscere i diversi ruoli e responsabilità in una review formale

FL-3.2.3 (K2) Spiegare le differenze tra i diversi tipi di review: review informale, walkthrough, review tecnica e ispezione

FL-3.2.4 (K3) Applicare una tecnica di review a un prodotto di lavoro per rilevare difetti

FL-3.2.5 (K2) Spiegare i fattori che contribuiscono a una review di successo

3.1 Fondamenti del Testing Statico

A differenza del testing dinamico, che richiede l'esecuzione del software da testare, il testing statico è basato sull'esame manuale dei prodotti di lavoro (cioè le review), oppure sulla valutazione del codice o di un prodotto di lavoro guidata da uno strumento software (cioè l’analisi statica). Entrambi i tipi di testing statico valutano il codice o un altro prodotto di lavoro senza eseguire effettivamente il codice o il prodotto di lavoro stesso.

L'analisi statica è importante per i sistemi safety-critical (come software avionico, medicale o nucleare), ma è diventata importante e comune anche in altre situazioni. Ad esempio, l'analisi statica è una parte importante del testing di sicurezza. L'analisi statica è spesso incorporata anche in strumenti di build e distribuzione del software in modo automatizzato, ad esempio il continuous delivery e il continuous deployment nello sviluppo Agile.

3.1.1 Prodotti di Lavoro che possono essere Esaminati dal Testing Statico

Quasi tutti i prodotti di lavoro possono essere esaminati attraverso il testing statico (review e/o analisi statica), ad esempio:

- Specifiche, che includono requisiti di business, requisiti funzionali e requisiti di sicurezza
- Epic, user story e criteri di accettazione
- Specifiche architetturali e di progettazione
- Codice
- Testware, che include Test Plan, test case, procedure di test e test script automatizzati
- Guide utente
- Pagine Web
- Contratti, piani di progetto, pianificazioni e budget
- Set up di Configurazione e dell’infrastruttura
- Modelli, come gli activity diagram, che possono essere utilizzati per il testing Model-Based (si veda il Syllabus ISTQB-CTFL-MBT e Kramer 2016)

Le review possono essere applicate a qualsiasi prodotto di lavoro che i partecipanti siano in grado di leggere e comprendere. L'analisi statica può essere applicata in modo efficiente a qualsiasi prodotto di lavoro con una struttura formale (in genere codice o modelli) per i quali esista uno strumento di analisi statica appropriato. L'analisi statica può essere svolta anche utilizzando strumenti che valutino i prodotti di lavoro scritti in linguaggio naturale come i requisiti (ad esempio, verifica dell’ortografia, grammatica e leggibilità).

3.1.2 Benefici del Testing Statico

Le tecniche di test statico offrono una serie di vantaggi. Il testing statico, se applicato nelle fasi iniziali del ciclo di vita dello sviluppo software, consente il rilevamento anticipato dei difetti, prima del testing dinamico (ad es. nelle review delle specifiche dei requisiti o della progettazione, nel raffinamento del backlog, ecc.). I difetti rilevati in anticipo sono spesso molto meno costosi da eliminare, rispetto ai difetti rilevati successivamente nel ciclo di vita, specialmente rispetto ai difetti rilevati dopo che il software è stato rilasciato ed è utilizzato. L’utilizzo di tecniche di test statico per rilevare difetti, e quindi correggerli tempestivamente, è quasi sempre molto più economico per l’organizzazione, rispetto all'utilizzo del testing dinamico per rilevare difetti nell’oggetto di test e quindi correggerli, specialmente quando si considerino i costi aggiuntivi associati all'aggiornamento di altri prodotti di lavoro e all'esecuzione di testing confermativo e di regression testing.

Ulteriori vantaggi del testing statico possono includere:

- Rilevare e correggere difetti in modo più efficiente, e prima dell'esecuzione dinamica dei test
- Identificare i difetti che non sono facilmente individuabili con il testing dinamico
- Prevenire difetti nella progettazione o nella codifica, scoprendo inconsistenze, ambiguità, contraddizioni, omissioni, inaccuratezze e ridondanze nei requisiti
- Incrementare la produttività dello sviluppo (ad es. grazie a una migliorata progettazione, a codice più manutenibile)
- Ridurre costi e tempi di sviluppo
- Ridurre costi e tempi del testing
- Ridurre il costo totale della qualità durante il ciclo vita del software, grazie a un minor numero di failure nelle fasi successive del ciclo di vita o dopo il rilascio in produzione
- Migliorare la comunicazione tra i membri del team durante la partecipazione a review

3.1.3 Differenze fra Testing Statico e Dinamico

Testing statico e testing dinamico possono avere gli stessi obiettivi (si veda il paragrafo 1.1.1), come ad esempio fornire una valutazione della qualità dei prodotti di lavoro e identificare i difetti il prima possibile. Testing statico e testing dinamico sono complementari tra loro nella rilevazione di diversi tipi di difetti.

Una differenza principale è che il testing statico rileva difetti direttamente nei prodotti di lavoro, piuttosto che identificare failure causati da difetti durante l'esecuzione del software. Un difetto può risiedere in un prodotto di lavoro per molto tempo senza causare alcun failure. Il percorso in cui si trova il difetto può essere esercitato raramente o essere difficile da raggiungere, quindi non sarà facile implementare ed eseguire un test dinamico che lo rilevi. Il testing statico può essere in grado di trovare il difetto con un effort minore.

Un'altra distinzione è che il testing statico può essere utilizzato per migliorare la consistenza e la qualità interna dei prodotti di lavoro, mentre il testing dinamico si concentra sui comportamenti visibili esternamente.

Confrontato con il testing dinamico, i difetti tipici che sono più facili e meno costosi da rilevare e correggere attraverso il testing statico includono:

- Difetti dei requisiti (ad es. inconsistenze, ambiguità, contraddizioni, omissioni, inaccuratezze e ridondanze)
- Difetti di progettazione (ad es. algoritmi o strutture di database inefficienti, alto accoppiamento, bassa coesione)
- Difetti di codifica (ad es. variabili con valori indefiniti, variabili dichiarate ma mai utilizzate, codice non raggiungibile, codice duplicato)
- Deviazioni dagli standard (ad es. mancanza di aderenza agli standard di codifica)
- Specifiche di interfaccia errate (ad es. differenti unità di misura utilizzate dal sistema chiamante rispetto al sistema chiamato)
- Vulnerability di sicurezza (ad es. suscettibilità ai buffer overflow)
- Lacune o inaccuratezza nella tracciabilità o nella copertura della base di test (ad es. test mancanti per un criterio di accettazione)

Inoltre, molti tipi di difetti di manutenibilità possono essere rilevati solo con il testing statico (ad es. modularizzazione non appropriata, scarsa riusabilità dei componenti, codice difficile da analizzare e modificare senza introdurre nuovi difetti).

3.2 Processo di Review

Le review variano da informali a formali. Le review informali sono caratterizzate dal non seguire un processo definito e dal non avere un output formale documentato. Le review formali sono caratterizzate dalla partecipazione del team, da risultati della review documentati e da procedure documentate per condurre la review. La formalità di un processo di review è legata a fattori come il modello del ciclo di vita dello sviluppo software, la maturità del processo di sviluppo, la complessità del prodotto di lavoro da sottoporre a review, requisiti normativi e legali, e/o la necessità di un audit trail.

Il focus di una review dipende dagli obiettivi concordati della review (ad es. trovare difetti, acquisire conoscenza, educare i partecipanti come tester e nuovi membri del team, oppure discutere e prendere decisioni attraverso un consenso).

Lo standard ISO (ISO/IEC 20246) contiene descrizioni più dettagliate e approfondite del processo di review per i prodotti di lavoro, includendo ruoli e tecniche di review.

3.2.1 Processo di Review dei Prodotti di Lavoro

Il processo di review comprende le seguenti attività principali:

Pianificazione

- Definire l'ambito, che include lo scopo della review, quali documenti o parti di documenti sottoporre a review e le caratteristiche di qualità da valutare
- Stimare effort e tempistiche
- Identificare le caratteristiche della review come il tipo di review, con ruoli, attività e checklist
- Selezionare le persone che partecipano alla review e assegnare i rispettivi ruoli
- Definire i criteri di ingresso e uscita per i tipi di review più formali (ad es., ispezioni)
- Verificare che i criteri di ingresso siano soddisfatti (per i tipi di review più formali)

Inizio della review

- Distribuire (fisicamente o per via elettronica) il prodotto di lavoro e altro materiale, come moduli di issue log, checklist e prodotti di lavoro correlati
- Spiegare ai partecipanti l’ambito, gli obiettivi, i processi, i ruoli e i prodotti di lavoro
- Rispondere a qualsiasi domanda che i partecipanti possano avere sulla review

Review individuale (o preparazione individuale)

- Svolgere la review di tutto o parte del prodotto di lavoro
- Annotare potenziali difetti, raccomandazioni e domande

Comunicazione e analisi dei problemi

- Comunicare i potenziali difetti identificati (ad es. in un review meeting)
- Analizzare i potenziali difetti, assegnando il relativo proprietario e stato
- Valutare e documentare le caratteristiche di qualità
- Valutare i risultati della review rispetto ai criteri di uscita, per prendere una decisione sulla review (prodotto rifiutato; che richiede importanti modifiche; accettato con eventuali modifiche minori)

Correzione e reporting

- Creare defect report per quei risultati della review che richiedono modifiche a un prodotto di lavoro

- Correggere i difetti rilevati (normalmente svolto dall'autore) nel prodotto di lavoro sottoposto a review
- Comunicare i difetti alla persona o al team appropriati (se trovati in un prodotto di lavoro correlato al prodotto di lavoro sotto review)
- Registrare lo stato aggiornato dei difetti (nelle review formali), includendo potenzialmente il consenso dell'autore della segnalazione
- Raccogliere metriche (per tipi di review più formali)
- Verificare che i criteri di uscita siano soddisfatti (per tipi di review più formali)
- Accettare il prodotto di lavoro quando vengono raggiunti i criteri di uscita

I risultati di una review del prodotto di lavoro variano in base al tipo e alla formalità della review, come descritto nel paragrafo 3.2.3.

3.2.2 Ruoli e Responsabilità in una Review Formale

Una tipica review formale prevede i seguenti ruoli:

Autore

- Crea il prodotto di lavoro sotto review
- Corregge i difetti nel prodotto di lavoro sotto review (se necessario)

Management

- È responsabile della pianificazione della review
- Prende decisioni sull'esecuzione delle review
- Assegna staff, budget e tempi
- Monitora l'efficacia costi/benefici
- Prende decisioni in caso di risultati inadeguati

Facilitatore (o moderatore)

- Garantisce l'esecuzione efficace dei review meeting (quando svolti)
- Fa da mediatore, se necessario, tra diversi punti di vista
- È spesso la persona da cui dipende il successo della review

Review Leader

- Assume la responsabilità complessiva della review
- Decide chi sarà coinvolto e organizza tempi e luoghi di svolgimento della review

Reviewer

- Possono essere esperti in materia, persone che lavorano al progetto, stakeholder interessati al prodotto di lavoro e/o individui con specifici background tecnici o di business
- Identificano potenziali difetti nel prodotto di lavoro sotto review
- Possono rappresentare diverse prospettive (ad es. tester, sviluppatore, utente, operatore, business analyst, esperto di usabilità, ecc.)

Scribe (o recorder)

- Raccoglie i potenziali difetti rilevati durante l'attività di review individuale
- Registra nuovi potenziali difetti, open point e decisioni emersi dal review meeting (quando svolto)

In alcuni tipi di review, una persona può svolgere più di un ruolo e le azioni associate a ciascun ruolo possono anche variare in base al tipo di review. Inoltre, con l’introduzione di strumenti a supporto del processo di review, come il logging dei difetti, degli open point e delle decisioni, spesso uno scribe può non essere necessario.

Sono inoltre possibili ruoli più dettagliati, come descritto nello standard ISO (ISO/IEC 20246).

3.2.3 Tipi di Review

Sebbene le review possano essere utilizzate per vari scopi, uno degli obiettivi principali è quello di scoprire i difetti. Tutti i tipi di review possono aiutare nella rilevazione dei difetti e la selezione del tipo di review dovrebbe basarsi, tra i criteri di selezione, sulle esigenze del progetto, sulle risorse disponibili, sul tipo di prodotto e i relativi rischi, sul dominio di business e la cultura aziendale.

Un singolo prodotto di lavoro può essere soggetto di più di un tipo di review. Se vengono usati più tipi di review, l'ordine di applicazione può variare. Ad esempio, una review informale può essere svolta prima di una review tecnica, per garantire che il prodotto di lavoro sia pronto per una review tecnica

I tipi di review descritti di seguito possono essere svolti come peer review, cioè svolte da colleghi dell’autore, qualificati per fare lo stesso lavoro.

I tipi di difetti rilevati in una review dipendono soprattutto dal prodotto di lavoro sotto review (si veda il paragrafo 3.1.3 per esempi di difetti che possono essere rilevati durante le review di differenti prodotti di lavoro, e Gilb 1993 per dettagli sulle ispezioni formali)

Le review possono essere classificate in base a differenti attributi. Di seguito sono elencate le quattro tipologie di review più comuni e gli attributi associati.

Review Informale (ad esempio buddy check, pairing, pair review)

- Obiettivo principale: rilevare potenziali difetti
- Possibili obiettivi aggiuntivi: generare nuove idee o soluzioni, risolvere rapidamente problemi minori
- Non basate su un processo formale (documentato)
- Possono non richiedere un review meeting
- Possono essere eseguite da un collega dell'autore (buddy check) o da più persone
- I risultati possono essere documentati
- L’efficacia dipende dai reviewer
- Uso facoltativo di checklist
- Comunemente utilizzate nello sviluppo Agile

Walkthrough

- Obiettivi principali: rilevare difetti, migliorare il prodotto software, considerare implementazioni alternative, valutare conformità a standard e specifiche
- Possibili obiettivi aggiuntivi: scambiarsi idee su tecniche o variazione di approcci, fare training ai partecipanti, raggiungere il consenso
- La preparazione individuale prima del review meeting è facoltativa
- Il review meeting è in genere guidato dall'autore del prodotto di lavoro
- Lo scribe (recorder) è obbligatorio
- L'uso di checklist è facoltativo
- Può assumere la forma di scenari, dry run o simulazioni

- Sono prodotti i log dei difetti potenziali e i review report
- Può variare da completamente informale a molto formale

Review Tecnica

- Obiettivi principali: ottenere consenso, individuare potenziali difetti
- Possibili obiettivi aggiuntivi: valutare la qualità e prendere confidenza con il prodotto di lavoro, generare nuove idee, motivare e consentire agli autori di migliorare i prodotti di lavoro futuri, considerare implementazioni alternative
- I reviewer dovrebbero essere colleghi tecnici dell'autore ed esperti tecnici nella stessa o in altre discipline
- La preparazione individuale prima del review meeting è obbligatoria
- Il review meeting è facoltativo, guidato idealmente da un facilitatore esperto (tipicamente non l'autore)
- Lo scribe è obbligatorio, possibilmente non l'autore
- L'uso di checklist è facoltativo
- Sono prodotti i i log dei difetti potenziali e i review report

Ispezione

- Obiettivo principale: individuare potenziali difetti, valutare la qualità e prendere confidenza del prodotto di lavoro, prevenire difetti simili futuri attraverso l'apprendimento dell'autore e la root cause analysis
- Possibili obiettivi aggiuntivi: motivare e consentire agli autori di migliorare i prodotti di lavoro futuri e il processo di sviluppo software, raggiungere il consenso
- Segue un processo definito con output formali documentati, basati su regole e checklist
- Utilizza ruoli chiaramente definiti, come quelli specificati nel paragrafo 3.2.2 che sono obbligatori, e può includere un lettore dedicato (che legge ad alta voce il prodotto di lavoro durante il review meeting, spesso parafrasando, cioè descrivendolo con parole proprie)
- La preparazione individuale prima del review meeting è obbligatoria
- I reviewer sono colleghi dell'autore o esperti di altre discipline che sono rilevanti per il prodotto di lavoro
- Vengono utilizzati criteri di ingresso e di uscita predefiniti
- Lo scribe è obbligatorio
- Il review meeting è guidato da un facilitatore esperto (non dall'autore)
- L'autore non può ricoprire il ruolo di review leader, lettore o scribe
- Vengono prodotti i log dei potenziali difetti e i review report
- Sono raccolte metriche, utilizzate per migliorare l'intero processo di sviluppo software, incluso il processo di ispezione

3.2.4 Applicare Tecniche di Review

Esistono numerose tecniche di review che possono essere applicate durante l’attività di review individuale (preparazione individuale) per scoprire i difetti. Queste tecniche possono essere utilizzate in tutti i tipi di review sopra descritti. L'efficacia delle tecniche può variare a seconda del tipo di review adottata. Esempi di diverse tecniche di review individuale per vari tipi di review sono elencati di seguito.

Ad hoc

In una review ad hoc, i reviewer sono forniti di una guida limitata (o nessuna) su come dovrebbe essere eseguita. I reviewer spesso leggono il prodotto di lavoro sequenzialmente, identificando e documentando i problemi man mano che li incontrano. La review ad hoc è una tecnica comunemente utilizzata che richiede poca preparazione. Questa tecnica dipende fortemente dalle competenze del reviewer e può portare a numerose segnalazioni duplicate di problemi da parte di reviewer differenti.

Checklist-based

Una review checklist-based è una tecnica sistematica, in cui i reviewer rilevano problemi sulla base di checklist distribuite all'inizio della review (ad es. dal facilitatore). Una checklist di una review è composta da un insieme di domande basate su difetti potenziali, che possono essere derivati dall'esperienza. Le checklist dovrebbero essere specifiche per il tipo di prodotto di lavoro in esame e dovrebbero essere regolarmente manutenute per coprire i tipi di problemi non rilevati in precedenti review. Il vantaggio principale della tecnica checklist-based è la copertura sistematica di tipologie di difetti tipici. Durante la review individuale bisogna fare attenzione a non seguire solo la checklist, ma cercare anche difetti al di fuori della checklist stessa.

Scenari e dry run

In una review scenario-based, i reviewer sono forniti di linee guida strutturate su come leggere il prodotto di lavoro. Una review scenario-based supporta i reviewer nell’esecuzione di “dry run” sul prodotto di lavoro in base all’utilizzo atteso del prodotto di lavoro stesso (se il prodotto di lavoro è documentato in un formato adeguato, come gli use case). Questi scenari forniscono ai reviewer delle linee guida migliori su come identificare specifici tipi di difetti, rispetto al semplice elenco in una checklist. Come nelle review checklist-based, per non trascurare altri tipi di difetti (ad es. funzionalità mancanti), i reviewer non dovrebbero essere vincolati agli scenari documentati.

Perspective-based

Nella lettura perspective-based, analogamente a una review role-based, i reviewer, durante le review individuali, assumono i punti di vista di differenti stakeholder. Tipici punti di vista di stakeholder includono utenti finali, marketing, progettisti, tester o esercizio (operations). L'utilizzo dei differenti punti di vista degli stakeholder porta a un maggiore approfondimento nella review individuale con meno duplicazioni dei problemi rilevati dai reviewer.

Inoltre, la lettura perspective-based richiede anche che i reviewer cerchino di utilizzare il prodotto di lavoro sotto review per generare il prodotto che deriverebbero da esso. Ad esempio, un tester, se esegue una lettura perspective-based di una specifica dei requisiti, potrebbe tentare di generare un draftdei test di accettazione per verificare se sono state incluse tutte le informazioni necessarie. Infine, nella lettura perspective-based ci si aspetta che vengano utilizzate le checklist.

Studi empirici hanno mostrato che la lettura perspective-based è la tecnica generale più efficace per la review dei requisiti e di prodotti di lavoro tecnici. Un fattore chiave di successo è includere e pesare diversi punti di vista degli stakeholder in modo appropriato, basandosi sui rischi. Si veda Shul 2000 per dettagli sulla lettura perspective-based e Sauer 2000 per l'efficacia delle differenti techniche di review.

Role-based

Una review role-based è una tecnica in cui i reviewer valutano il prodotto di lavoro dalla prospettiva dei ruoli di singoli stakeholder. I ruoli tipici includono specifiche tipologie di utenti finali (esperti, inesperti, senior, junior, ecc.) e ruoli specifici nell'organizzazione (user administrator, system administrator, performance tester, ecc.). Si applicano gli stessi principi della lettura prospective-based perché i ruoli sono simili.

3.2.5 Fattori di Successo per le Review

Per ottenere una review di successo, devono essere considerati un tipo appropriato di review e le tecniche utilizzate. Inoltre, esistono altri fattori che influenzano il risultato della review.

I fattori di successo per le review di tipo organizzativo includono:

- Ogni review ha obiettivi chiari, definiti durante la pianificazione della review e utilizzati come criteri di uscita misurabili

- Sono applicati i tipi di review che sono adatti a raggiungere gli obiettivi e sono appropriati al tipo e al livello dei prodotti di lavoro software e ai partecipanti
- Qualsiasi tecnica di review utilizzata, come la review checklist-based o role-based, è adatta all’efficace identificazione dei difetti nel prodotto di lavoro sotto review
- Qualsiasi checklist utilizzata indirizza i principali rischi ed è sempre aggiornata
- I documenti di grandi dimensioni sono redatti e sottoposti a review in piccole parti, in modo che il controllo della qualità venga svolto fornendo agli autori feedback anticipati e frequesti sui difetti
- I partecipanti hanno il tempo sufficiente per prepararsi
- Le review sono schedulate con adeguato preavviso
- Il management supporta il processo di review (ad es. considerando un tempo adeguato per le attività di review nella schedulazione di progetto)
- Le review sono integrate nella politica del test e nelle procedure di qualità dell’organizzazione.

I fattori di successo per le review relativi alle persone includono:

- Sono coinvolte le persone giuste per raggiungere gli obiettivi della review, ad esempio, le persone con differenti competenze o punti di vista, che possono utilizzare il documento come input del proprio lavoro
- I tester sono considerati dei reviewer validi che contribuiscono alla review e imparano sul prodotto di lavoro, consentendo loro di preparare test più efficaci e in anticipo
- I partecipanti dedicano tempo e attenzione ai dettagli
- Le review sono condotte a piccoli blocchi, in modo che i reviewer non perdano la concentrazione durante la review individuale e/o durante il review meeting (quando svolto)
- I difetti rilevati sono riconosciuti, apprezzati e gestiti in modo oggettivo
- Il meeting è ben gestito, in modo che i partecipanti lo considerino tempo utile
- La review è condotta in un'atmosfera di fiducia; il risultato non sarà utilizzato per la valutazione dei partecipanti
- I partecipanti evitano il linguaggio del corpo e comportamenti che potrebbero indicare noia, esasperazione o ostilità verso altri partecipanti
- E’ fornita una formazione adeguata, specialmente per i tipi di review più formali come le ispezioni
- Viene promossa una cultura dell'apprendimento e del miglioramento dei processi

(Si veda Gilb 1993, Wiegers 2002 e van Veenendaal 2004 per ulteriori informazioni sulle review di successo.)

4. Tecniche di Test – 330 minuti

Parole Chiave

analisi ai valori limite, copertura, copertura delle decisioni, copertura delle istruzioni, error guessing, partizionamento di equivalenza, tecnica di test, tecnica di test basata sull'esperienza, tecnica di test black-box, tecnica di test white-box, testing checklist-based, testing della tabella delle decisioni, testing degli use case, testing delle transizioni di stato, testing esplorativo,

Obiettivi di Apprendimento per le Tecniche di Test

4.1 Categorie di Tecniche di Test

FL-4.1.1 (K2) Spiegare le caratteristiche, i punti in comune e le differenze tra le tecniche di test black-box, tecniche di test white-box e tecniche di test basata sull'esperienza

4.2 Tecniche di Test Black-box

FL-4.2.1 (K3) Applicare il partizionamento di equivalenza per derivare test case dai requisiti specificati

FL-4.2.2 (K3) Applicare l'analisi ai valori limite per derivare test case dai requisiti specificati

FL-4.2.3 (K3) Applicare il testing della tabella delle decisioni per derivare test case dai requisiti specificati

FL-4.2.4 (K3) Applicare il testing delle transizioni di stato per derivare test case dai requisiti specificati

FL-4.2.5 (K2) Spiegare come derivare test case da uno use case

4.3 Tecniche di Test White-box

FL-4.3.1 (K2) Spiegare la copertura delle istruzioni

FL-4.3.2 (K2) Spiegare la copertura delle decisioni

FL-4.3.3 (K2) Spiegare il valore della copertura delle istruzioni e delle decisioni

4.4 Tecniche di Test Basate sull’Esperienza

FL-4.4.1 (K2) Spiegare l'error guessing

FL-4.4.2 (K2) Spiegare il testing esplorativo

4.1 FL-4.4.3 (K2) Spiegare il testing checklist-basedCategorie di Tecniche di Test

Lo scopo di una tecnica di test, incluse quelle discusse in questo paragrafo, è di aiutare a identificare le condizioni di test, i test case e i dati di test.

La scelta delle tecniche di test da utilizzare dipende da una serie di fattori, che includono:

- Complessità del componente o sistema

- Standard normativi
- Requisiti del cliente o contrattuali
- Livelli e tipi di rischio
- Documentazione disponibile
- Conoscenza e competenze dei tester
- Strumenti disponibili
- Tempi e budget
- Modello del ciclo di vita dello sviluppo software
- Tipi di difetti attesi nel componente o sistema

Alcune tecniche sono più applicabili in determinate situazioni e a certi livelli di test, altre sono applicabili a tutti i livelli di test. Durante la creazione dei test case, i tester usano generalmente una combinazione di tecniche di test per ottenere i migliori risultati in termini di effort del test.

L'uso delle tecniche di test durante le attività di analisi, progettazione e implementazione dei test può variare da molto informale (poca o nessuna documentazione) a molto formale. Il livello appropriato di formalità dipende dal contesto del test, inclusi la maturità dei processi di test e di sviluppo, i vincoli temporali, i requisiti normativi o di safety, la conoscenza e le competenze delle persone coinvolte, e il modello del ciclo di vita dello sviluppo software adottato.

4.1.1 Categorie di Tecniche di Test e loro Caratteristiche

In questo Syllabus le tecniche di test sono classificate come black-box, white-box o basate sull'esperienza.

Le tecniche di test black-box (chiamate anche tecniche comportamentali o basate sul comportamento) si basano su un’analisi della base di test appropriata (ad es. documenti dei requisiti formali, specifiche, use case, user story o processi di business). Queste tecniche sono applicabili sia al testing funzionale che non-funzionale. Le tecniche di test black-box si concentrano sugli input e output dell'oggetto di test, senza riferimenti alla sua struttura interna.

Le tecniche di test white-box (chiamate anche tecniche strutturali o basate sulla struttura) si basano su un'analisi dell'architettura, della progettazione di dettaglio, della struttura interna o del codice dell'oggetto di test. A differenza delle tecniche di test black-box, le tecniche di test white-box si concentrano sulla struttura e sull'elaborazione interna dell'oggetto di test.

Le tecniche di test basata sull'esperienza sfruttano l'esperienza di sviluppatori, tester e utenti per progettare, implementare ed eseguire i test. Queste tecniche sono spesso combinate con tecniche di test black-box e white-box.

Caratteristiche comuni alle tecniche di test black-box includono:

- Condizioni di test, test case e dati di test sono derivati da una base di test, che può includere requisiti software, specifiche, use case e user story
- I test case possono essere utilizzati per rilevare discrepanze tra i requisiti e la loro implementazione, nonché deviazioni dai requisiti
- La copertura viene misurata in base agli elementi testati nella base di test e alla tecnica applicata alla base di test

Caratteristiche comuni alle tecniche di test white-box includono:

- Condizioni di test, test case e dati di test sono derivati da una base di test che può includere codice, architettura software, progettazione di dettaglio o qualsiasi altra fonte di informazione relativa alla struttura del software

- La copertura viene misurata in base agli elementi testati all'interno di una struttura selezionata (ad es. il codice o le interfacce) e alla tecnica applicata alla base di test

Caratteristiche comuni alle tecniche di test basate sull'esperienza includono:

- Condizioni di test, test case e dati di test sono derivati da una base di test che può includere la conoscenza e l’esperienza di tester, sviluppatori, utenti e altri stakeholder

La conoscenza e l’esperienza includono l'uso previsto del software, il suo ambiente, i difetti probabili e la distribuzione di questi difetti.

Lo standard internazionale (ISO/IEC/IEEE 29119-4) contiene descrizioni delle tecniche di test e delle misure di copertura corrispondenti (per maggiori informazioni sulle tecniche si veda Craig 2002 e Copeland 2004).

4.2 Tecniche di Test Black-box

4.2.1 Partizionamento di Equivalenza

Il partizionamento di equivalenza divide i dati in partizioni (chiamate anche classi di equivalenza) basandosi sull’assunzione che tutti gli elementi di una determinata partizione vengano elaborati allo stesso modo (si veda Kaner 2013 e Jorgensen 2014). Esistono partizioni di equivalenza per valori validi e valori invalidi.

- I valori validi sono valori che dovrebbero essere accettati dal componente o sistema. Una partizione di equivalenza che contiene valori validi è chiamata "partizione di equivalenza valida".
- I valori invalidi sono valori che dovrebbero essere rifiutati dal componente o sistema. Una partizione di equivalenza che contiene valori invalidi è chiamata "partizione di equivalenza invalida".
- Le partizioni possono essere identificate considerando qualsiasi parametro di dati relativo all'oggetto di test, inclusi input, output, valori interni, valori temporali (ad es. prima o dopo un evento) e parametri di interfaccia (ad es. componenti integrati sottoposti a test durante il testing di integrazione).
- Qualsiasi partizione può essere suddivisa, se necessario, in sotto-partizioni.
- Ogni valore deve appartenere a una e una sola partizione di equivalenza.
- Quando le partizioni di equivalenza invalide sono utilizzate nei test case, dovrebbero essere testate singolarmente, cioè non combinate con altre partizioni di equivalenza invalide, per garantire che i failure non siano mascherati (effetto di defect masking). I failure possono essere mascherati quando diversi failure accadono contemporaneamente ma soltanto un failure è visibile, impedendo di rilevare gli altri failure.

Per ottenere una copertura del 100% con questa tecnica, i test case devono coprire tutte le partizioni identificate (incluse le partizioni invalide) utilizzando almeno un valore di ciascuna partizione. La copertura viene misurata come numero di partizioni di equivalenza testate da almeno un valore, diviso per il numero totale di partizioni di equivalenza identificate, normalmente espressa in percentuale. Il partizionamento di equivalenza è applicabile a tutti i livelli di test.

4.2.2 Analisi ai Valori Limite

L'analisi ai valori limite (BVA = Boundary Value Analysis) è un'estensione del partizionamento di equivalenza, ma può essere utilizzata solo quando la partizione è ordinata, costituita da dati numerici o sequenziali. I valori minimo e massimo (o primo e ultimo valore) di una partizione sono i suoi valori limite (si veda Beizer 1990).

Ad esempio, si supponga che un campo di input accetti in input un singolo valore numerico intero e che si utilizzi una Interfaccia Utente tale che non siano possibili input numerici non interi. L'intervallo valido ha valori da 1 a 5 inclusi. Esistono quindi tre partizioni di equivalenza: invalida (valore troppo basso); valida; invalida (valore troppo alto). Per la partizione di equivalenza valida, i valori limite sono 1 e 5. Per la partizione invalida (valore troppo alto), il valori limite è 6. Per la partizione invalida (valore troppo basso), esiste solo il valore limite 0, perché la partizione contiene un solo elemento.

Nell'esempio sopra descritto si identificano due valori limite per ogni limite. Il limite tra partizione invalida (valore troppo basso) e partizione valida fornisce i valori di test 0 e 1. Il limite tra partizione valida e partizione invalida (valore troppo alto) fornisce valori di test 5 e 6. Alcune varianti di questa tecnica identificano tre valori limite per ogni limite: i valori appena prima, sul e appena oltre il limite. Nell'esempio precedente, utilizzando i valori limite a tre valori, i valori limite inferiori sono 0, 1 e 2, mentre i valori limite superiori sono 4, 5, e 6 (si veda Jorgensen 2014).

Il comportamento ai limiti delle partizioni di equivalenza è più probabile che sia errato rispetto al comportamento all'interno delle partizioni. È importante ricordare che sia i limiti specificati nella documentazione sia quelli implementati possono essere erroneamente spostati a posizioni al di sopra o al di sotto delle loro posizioni previste, possono essere del tutto omessi o possono essere integrati con limiti aggiuntivi indesiderati. Il testing e l’analisi ai valore limite possono rivelare quasi tutti questi difetti ai limiti, forzando il software a mostrare comportamenti di una partizione differenti da quella a cui il valore limite dovrebbe appartenere.

L'analisi ai valori limite può essere applicata a tutti i livelli di test. Questa tecnica è generalmente utilizzata per testare requisiti che richiedono un intervallo di valori numerici (inclusi date e orari). La copertura dei valori limite viene misurata come il numero di valori limite testati, diviso per il numero totale di valori limite identificati, normalmente espressa in percentuale.

4.2.3 Testing della Tabella delle Decisioni

Le tabelle delle decisioni sono un approccio valido per descrivere regole di business complesse che un sistema deve implementare. Nel creare le tabelle delle decisioni, il tester identifica le condizioni (spesso gli input) e le azioni risultanti (spesso gli output) del sistema. Questi elementi formano le righe della tabella, generalmente con le condizioni nella parte superiore e le azioni nella parte inferiore della tabella. Ogni colonna corrisponde a una regola decisionale, definita da una combinazione unica di condizioni che determinano l'esecuzione delle azioni associate a tale regola. I valori delle condizioni e delle azioni vengono normalmente indicati con valori Booleani (vero o falso) o valori discreti (ad es. rosso, verde, blu), ma possono anche essere numeri o intervalli numerici. Questi diversi tipi di notazioni delle condizioni e azioni potrebbero essere utilizzati insieme nella stessa tabella.

La notazione comune nelle tabelle delle decisioni è la seguente:

Per le condizioni:

- “Y” indica che la condizione è vera (può anche essere indicato come “T” o “1”)
- “N” indica che la condizione è falsa (può anche essere indicato come “F” o “0”)
- “-“ indica che il valore della condizione non è significativo (può anche essere indicato come “N/A”)

Per le azioni:

- “X” indica che l'azione dovrebbe accadere (può anche essere indicato come “Y” o “T” o “1”)
- Campo vuoto indica che l'azione non dovrebbe accadere (può anche essere indicato come “-“ o “N” o “F” o “0”)

Una tabella delle decisioni completa ha abbastanza colonne (test case) per coprire ogni combinazione di condizioni. Cancellando colonne che non influenzano il risultato, il numero di test case possono ridursi in modo considerevole. Ad esempio, eliminando le colonne contenenti combinazioni di condizioni impossibili. Per ulteriori informazioni su come collassare le tabelle delle decisioni, si veda ISTQB-CTAL-TA.

La tipica copertura minima standard per il testing della tabella delle decisioni è avere almeno un test case per ogni regola decisionale nella tabella. Questo in genere implica la copertura di tutte le combinazioni di condizioni. La copertura è misurata come il numero di regole decisionali testate da almeno un test case, diviso per il numero totale di regole decisionali, normalmente espressa in percentuale.

La forza del testing della tabella delle decisioni è aiutare a identificare tutte le combinazioni importanti di condizioni, alcune delle quali potrebbero altrimenti essere trascurate. Aiuta anche a trovare eventuali lacune nei requisiti. Può essere applicato a tutte le situazioni in cui il comportamento del software dipende da una combinazione di condizioni, a qualsiasi livello di test.

4.2.4 Testing delle Transizioni di Stato

Componenti o sistemi possono rispondere in modo diverso a un evento in base alle condizioni attuali o alla storia precedente (ad es. gli eventi che si sono verificati da quando il sistema è stato inizializzato). La storia precedente può essere riassunta usando il concetto di stati. Un diagramma delle transizioni di stato mostra i possibili stati del software, nonché come il software entra, esce e transita da uno stato all'altro. Una transizione è innescata da un evento (ad es. l'inserimento da parte dell’utente di un valore di input in un campo). L'evento genera una transizione. Lo stesso evento può generare due o più transizioni differenti dallo stesso stato. Il cambiamento di stato (state change) può generare un'azione nel software (ad es. fornire in output un calcolo o un messaggio di errore).

Una tabella delle transizioni di stato mostra tutte le transizioni valide e potenzialmente invalide tra gli stati, così come gli eventi, e le azioni risultanti dalle transizioni valide. Gli state transtion diagram mostrano normalmente solo le transizioni valide ed escludono le transizioni invalide.

I test possono essere progettati per coprire una sequenza tipica di stati, per esercitare tutti gli stati, per esercitare ogni transizione, per esercitare specifiche sequenze di transizioni o per testare transizioni invalide.

Il testing della transizioni di stato viene utilizzato per gli applicativi basati su menu ed è ampiamente utilizzato negli applicativi software embedded. La tecnica è adatta anche per la modellazione di uno scenario di business con stati specifici, o per testare la navigazione delle schermate. Il concetto di stato è astratto, poiché può rappresentare poche righe di codice o un intero processo di business.

La copertura viene comunemente misurata come il numero di stati o di transizioni identificati e testati diviso per il numero totale di stati o di transizioni identificati nell'oggetto di test, espressa normalmente in percentuale. Per maggiori informazioni sui criteri di copertura per il testing delle transizioni di stato, si veda ISTQB-CTAL-TA.

4.2.5 Testing degli Use Case

I test possono essere derivati da use case, che sono un modo specifico di progettare le interazioni con elementi software. Gli use case incorporano i requisiti per le funzioni software. Gli use case sono associati ad attori (utenti umani, hardware esterno, altri componenti o sistemi) e soggetti (il componente o il sistema a cui lo use case è applicato).

Ogni use case specifica un comportamento che un soggetto può eseguire in collaborazione con uno o più attori (UML 2.5.1 2017). Uno use case può essere descritto da interazioni e da attività, da precondizioni, postcondizioni e linguaggio naturale ove appropriato. Le interazioni tra gli attori e il soggetto possono generare modifiche allo stato del soggetto. Le interazioni possono essere rappresentate graficamente da workflow, activity diagram o modelli di processi di business.

Uno use case può includere possibili variazioni del comportamento principale, includendo comportamenti alternativi o le eccezioni per la gestione degli errori, (per descrivere la risposta del sistema e il ripristino da errori di codifica, errori applicativi e di comunicazione, che generano per esempio un messaggio di errore). I test sono progettati per esercitare i comportamenti definiti nello use case (principale, eccezione o alternativo). La copertura può essere misurata come numero di comportamenti dello use case testati, diviso per il numero totale di comportamenti dello use case, normalmente espressa in percentuale.

Per ulteriori informazioni sui criteri di copertura per il testing degli use case, si veda ISTQB-CTAL-TA.

4.3 Tecniche di Test White-box

Il testing white-box si basa sulla struttura interna dell'oggetto di test. Le tecniche di test white-box possono essere utilizzate a tutti i livelli di test, ma le due tecniche relative al codice discusse in questo paragrafo sono più comunemente utilizzate a livello di test di componente. Esistono tecniche più avanzate che vengono utilizzate in alcuni ambienti safety-critical, mission-critical o high-integrity per ottenere una copertura maggiore e più forte, ma queste tecniche non sono trattate in questo Syllabus. Per ulteriori informazioni su tali tecniche, si veda ISTQB-CTAL-TTA.

4.3.1 Testing e Copertura delle Istruzioni

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

4.3.2 Testing e Copertura delle Decisioni

Il testing delle decisioni esercita le decisioni nel codice e testa il codice che viene eseguito in base agli esiti decisionali. Per fare questo, i test case seguono il flusso di controllo (control flow) che parte da un punto decisionale (ad es. per un'istruzione IF, esiste un test case per l’esito vero e un test case per l’esito falso; per un'istruzione CASE sarebbero necessari test case per tutti gli esiti possibili, incluso l’esito di default).

La copertura viene misurata come il numero di esiti decisionali eseguiti dai test diviso per il numero totale di esiti decisionali nell'oggetto di test, normalmente espressa in percentuale.

4.3.3 Il Valore del Testing delle Istruzioni e delle Decisioni

Il raggiungimento del 100% di copertura delle istruzioni, assicura che tutte le istruzioni eseguibili nel codice siano state testate almeno una volta, ma non garantisce che tutta la logica decisionale sia stata testata. Tra le due tecniche white-box discusse in questo Syllabus, il testing delle istruzioni può fornire una copertura minore rispetto al testing delle decisioni.

Il raggiungimento del 100% di copertura delle decisioni, assicura l’esecuzione di tutti gli esiti decisionali, che includono il test dell’esito vero e anche il test dell’esito falso, anche quando non vi è alcuna esplicita istruzione associata all’esito falso (ad es. un’istruzione di IF senza un ELSE nel codice). La copertura delle istruzioni aiuta a trovare difetti nel codice che non era stato esercitato da altri test. La copertura delle decisioni aiuta a rilevare difetti nel codice quando altri test non hanno eseguito entrambi i percorsi relativi agli esiti vero e falso.

Il raggiungimento del 100% di copertura delle decisioni garantisce il 100% di copertura delle istruzioni (ma non viceversa).

4.4 Tecniche di Test Basate sull’Esperienza

Quando si applicano tecniche di test basate sull'esperienza, i test case sono derivati dalla competenza e dall’intuizione del tester, e dalla sua esperienza con applicazioni e tecnologie similari. Queste tecniche possono essere utili nell'identificare test case che non sono facilmente identificabili da altre tecniche più sistematiche. In base all’approccio e all’esperienza del tester, queste tecniche possono raggiungere livelli di copertura ed efficacia molto variabili. La copertura con queste tecniche può essere difficile da valutare e potrebbe non essere misurabile.

Le tecniche basate sull'esperienza comunemente utilizzate sono discusse nei seguenti paragrafi.

4.4.1 Error Guessing

Error guessing è una tecnica utilizzata per anticipare il verificarsi di errori, difetti e failure, in base alla conoscenza del tester, che include:

- Come l'applicazione ha funzionato in passato
- Quali tipi di errori tendono ad essere commessi
- I failure che si sono verificate in altre applicazioni

Un approccio metodico alla tecnica di error guessing è creare una lista di possibili errori, difetti e failure, e progettare i test che genereranno tali failure e i difetti che le hanno causate. Queste liste di errori, difetti e failure possono essere create sulla base dell'esperienza, dei dati su difetti e failure, o di conoscenze diffuse del perché il software fallisce.

4.4.2 Testing Esplorativo

Nel testing esplorativo, i test informali (non predefiniti) sono progettati, eseguiti, registrati e valutati dinamicamente durante l'esecuzione dei test. I risultati dei test vengono utilizzati per conoscere meglio il componente o sistema, e per creare test relativi ad aree che possono richiedere maggiore testing.

Per strutturare l’attività, il testing esplorativo può essere condotto utilizzando il testing session-based. Nel testing session-based, il testing esplorativo viene svolto in un intervallo di tempo time-box, e il tester usa un Test Charter, che contiene gli obiettivi di test per guidare il testing. Il tester può utilizzare dei session sheet per documentare i passi eseguiti e le scoperte fatte durante la sessione.

Il testing esplorativo è più utile quando esistono poche specifiche o specifiche inadeguate, oppure in caso di pressione sui tempi di test. Il testing esplorativo è utile anche per integrare altre tecniche di test più formali.

Il testing esplorativo è fortemente associato a strategie di test reattive (si veda il paragrafo 5.2.2). Il testing esplorativo può includere l'uso di altre tecniche black-box, white-box e basate sull'esperienza.

4.4.3 Testing Checklist-Based

Nel testing checklist-based, i tester progettano, implementano ed eseguono test per coprire le condizioni di test riportate in una checklist. I tester, come parte dell'analisi, creano una nuova checklist oppure ne espandono una esistente, ma possono anche utilizzare una checklist esistente senza modifiche. Tali checklist possono essere create sulla base dell’esperienza, della conoscenza di quello che è importante per l'utente, o sulla comprensione del perché e di come il software fallisca.

Le checklist possono essere create per supportare vari tipi di test, inclusi il testing funzionale e non-funzionale. In assenza di test case dettagliati, il testing checklist-based può fornire linee guida e un certo grado di consistenza. Poiché sono liste di alto livello, è probabile che si verifichi una certa variabilità nei test effettivamente eseguiti, che potrebbe portare ad una copertura potenzialmente maggiore ma una minore ripetibilità.

5. Test Management (Gestione del Test) – 225 minuti

Parole Chiave

Approccio del test, Configuration Management (gestione della configurazione), controllo del test, criteri di ingresso, criteri di uscita, Defect Management (gestione dei difetti), defect report, livello di monitoraggio dei test, pianificazione dei test, rischio, rischio, rischio di prodotto, rischio di progetto, stima del test, strategia di test, Test Manager, Test Plan (Piano di Test), Test Progress Report, Test Summary Report, testing basato sul rischio, tester

Obiettivi di Apprendimento per il Test Management

5.1 Organizzazione del Test

FL-5.1.1 (K2) Spiegare i benefici e gli svantaggi del testing indipendente

FL-5.1.2 (K1) Identificare i compiti di un Test Manager e di un tester

5.2 Pianificazione e Stima dei Test

FL-5.2.1 (K2) Riassumere lo scopo e il contenuto di un Test Plan

FL-5.2.2 (K2) Distinguere tra diverse strategie di test

FL-5.2.3 (K2) Fornire esempi di potenziali criteri di ingresso e uscita

FL-5.2.4 (K3) Applicare la conoscenza della prioritizzazione e delle dipendenze tecniche e logiche, per schedulare l’esecuzione dei test per un determinato insieme di test case

FL-5.2.5 (K1) Identificare i fattori che influenzano l’effort relativo al testing

FL-5.2.6 (K2) Spiegare la differenza tra due tecniche di stima: la tecnica metrics-based e la tecnica expert-based

5.3 Monitoraggio e Controllo dei Test

FL-5.3.1 (K1) Ricordare le metriche utilizzate per il testing

FL-5.3.2 (K2) Riassumere gli obiettivi, i contenuti e i destinatari dei test report

5.4 Configuration Management (Gestione della Configurazione)

FL-5.4.1 (K2) Riassumere come il Configuration Management supporta il testing

5.5 Rischi e Testing

FL-5.5.1 (K1) Definire il livello di rischio utilizzando probabilità e impatto

FL-5.5.2 (K2) Distinguere tra rischi di progetto e rischi di prodotto

FL-5.5.3 (K2) Descrivere, utilizzando esempi, come l'analisi dei rischi di prodotto possa influenzare l'accuratezza e lo scopo del testing

5.6 Defect Management (Gestione dei Difetti)

FL-5.6.1 (K3) Scrivere un defect report che copra un difetto rilevato durante il testing

5.1 Organizzazione del Test

5.1.1 Testing Indipendente

Le attività di test possono essere svolte da persone con uno specifico ruolo nel testing o da persone con un altro ruolo (ad es. clienti). Un certo grado di indipendenza rende spesso il tester più efficace nel rilevare difetti, a causa delle differenze tra i cognitive bias dell'autore e del tester (si veda il paragrafo 1.5). L’indipendenza non sostituisce, comunque, la familiarità; e gli sviluppatori possono trovare in modo efficiente molti difetti nel proprio codice.

I gradi di indipendenza nel testing includono (dal più basso al più alto livello di indipendenza):

- Nessun tester indipendente; l'unica forma di test disponibile è quella degli sviluppatori che testano il proprio codice
- Sviluppatori o tester indipendenti all'interno dei team di sviluppo o del team di progetto; in questo caso potrebbero esserci sviluppatori che testano i prodotti dei loro colleghi
- Un team di test indipendente o un gruppo all'interno dell'organizzazione, che riporta gerarchicamente al project management o all’executive management
- Tester indipendenti dell'organizzazione di business o della comunità di utenti, o specializzati in specifici tipi di test, come l’usabilità, la sicurezza, l’efficienza delle prestazioni, la conformità a normative o la portabilità
- Tester indipendenti esterni all'organizzazione, che lavorano nella stessa sede (insourcing) o in una sede differente (outsourcing).

Per molti tipi di progetti, di solito è meglio avere livelli multipli di test, con alcuni di questi livelli gestiti da tester indipendenti. Gli sviluppatori dovrebbero partecipare al testing, specialmente ai livelli più bassi, in modo da esercitare il controllo sulla qualità del proprio lavoro.

Il modo in cui viene implementata l'indipendenza del testing varia a seconda del modello del ciclo di vita dello sviluppo software. Ad esempio, nello sviluppo Agile, i tester possono far parte di un team di sviluppo. In alcune organizzazioni che utilizzano metodi Agile, questi tester possono essere considerati membri di un più ampio team di test indipendente. In aggiunta, in queste organizzazioni, i Product Owner possono eseguire il testing di accettazione per validare le user story al termine di ogni iterazione.

I potenziali benefici del testing indipendente includono:

- E’ probabile che tester indipendenti riconoscano tipi di failure differenti rispetto agli sviluppatori, grazie aidifferenti background, le prospettive tecniche e i pregiudizi
- Un tester indipendente può verificare, sfidare o confutare le assunzioni formulate dagli stakeholder durante la specifica e l'implementazione del sistema
- Tester indipendenti di un fornitore possono fornire riscontri in modo oggettivo e obiettivo sul sistema sotto test, senza pressioni (politiche) da parte dell'azienda che li ha assunti

I potenziali svantaggi del testing indipendente includono:

- L’isolamento dal team di sviluppo, che può portare a mancanza di collaborazione, ritardi nel fornire feedback al team di sviluppo, o relazioni conflittuali con il team di sviluppo
- Gli sviluppatori possono perdere il senso di responsabilità per la qualità
- I tester indipendenti possono essere visti come un collo di bottiglia
- I tester indipendenti possono non ricevere alcune informazioni importanti (ad es. sull'oggetto di test).

Molte organizzazioni sono in grado di raggiungere con successo i benefici dell'indipendenza del test, evitando gli svantaggi.

5.1.2 Compiti di un Test Manager e di un Tester

In questo Syllabus vengono coperti due ruoli: il Test Manager e il tester. I compiti e le attività svolte da questi due ruoli dipendono dal contesto del progetto e del prodotto, dalle competenze delle persone nei rispettivi ruoli, e dall’organizzazione.

Il Test Manager ha la responsabilità generale del processo di test e di una leadership di successo delle attività di test. Il ruolo del Test Manager potrebbe essere svolto da un Test Manager professionista, da un Project Manager, da un Development Manager o da un Quality Assurance Manager. In progetti o organizzazioni più grandi, diversi team di test, ciascuno guidato da un Test Leader o Lead Tester, possono riferire a un Test Manager, test coach o test coordinator.

Le tipiche attività del Test Manager possono includere:

- Sviluppare o svolgere la review di una politica del test e di una strategia di test per l'organizzazione
- Pianificare le attività di test considerando il contesto e comprendendo obiettivi e rischi del test. Questo può includere la scelta degli approcci del test, la stima dei tempi di test, l'effort e i costi, l'acquisizione di risorse, la definizione dei livelli di test e dei cicli di test, e la pianificazione del defect management
- Redigere e aggiornare uno o più Test Plan
- Coordinare uno o più Test Plan con i Project Manager, Product Owner e altri
- Condividere la prospettiva del test con le altre attività di progetto, come la pianificazione dell'integrazione
- Avviare l'analisi, la progettazione, l'implementazione e l'esecuzione dei test, monitorare l’avanzamento e i risultati dei test, controllare lo stato dei criteri di uscita (o “definition of done”) e facilitare le attività di completamento dei test
- Preparare e rilasciare i Test Progress Report e i Test Summary Report in base alle informazioni raccolte
- Adattare la pianificazione in base ai risultati e all’avanzamento dei test (a volte documentati nei Test Progress Report e/o nei Test Summary Report, per i test già completati nel progetto) e intraprendere tutte le azioni necessarie per il controllo del test
- Supportare la predisposizione di un sistema di defect management e di un adeguato Configuration Management del testware
- Introdurre metriche adeguate per misurare l’avanzamento dei test e valutare la qualità del test e del prodotto
- Supportare la selezione e l'implementazione di strumenti per supportare il processo di test, includendo raccomandazioni sul budget per la selezione dello strumento (e possibilmente per l’acquisto e/o supporto), allocando tempi e risorse per i progetti pilota, e fornendo un supporto continuo all'uso degli strumenti
- Decidere sull'implementazione degli ambienti di test
- Promuovere e sostenere i tester, il team di test e la professione del test all'interno dell'organizzazione
- Sviluppare le competenze e le carriere dei tester (ad es. attraverso piani di formazione, valutazione delle prestazioni, coaching, ecc.)

Il modo con cui viene svolto il ruolo di Test Manager varia a seconda del ciclo di vita dello sviluppo software. Ad esempio, nello sviluppo Agile alcuni dei compiti sopra menzionati sono gestiti dal team Agile, in particolare quei compiti che riguardano le attività di test quotidiane, svolte spesso da un tester che lavora all'interno del team. Alcuni dei compiti che coinvolgono più team o l'intera organizzazione, oppure che hanno a che fare con la gestione delle persone, possono essere svolti da Test Manager al

di fuori del team di sviluppo, che a volte vengono chiamati Test Coach. Si veda Black 2009 per ulteriori informazioni sulla gestione del processo di test.

Le attività tipiche del tester possono includere:

- Svolgere la review e contribuire ai Test Plan
- Analizzare, eseguire la review e valutare requisiti, user story e criteri di accettazione, specifiche e modelli per la testabilità (cioè la base di test)
- Identificare e documentare le condizioni di test, e catturare la tracciabilità tra test case, condizioni di test e base di test
- Progettare, configurare e verificare gli ambienti di test, spesso coordinandosi con i gruppi di system administration e di network management (gestione della rete)
- Progettare e implementare test case e procedure di test
- Preparare e acquisire i dati di test
- Creare la schedulazione dettagliata dell'esecuzione dei test
- Eseguire i test, valutare i risultati e documentare le deviazioni dai risultati attesi
- Utilizzare gli strumenti appropriati per facilitare il processo di test
- Automatizzare i test in base alle esigenze (con l’eventuale supporto di uno sviluppatore o di un esperto di test automation)
- Valutare le caratteristiche non-funzionali, quali l’efficienza delle prestazioni, l’affidabilità, l’usabilità, la sicurezza, la compatibilità e la portabilità
- Eseguire la review di test sviluppati da altri

Le persone che lavorano all’analisi dei test, alla progettazione dei test, a specifici tipi di test o al test automation, possono essere specialisti in questi ruoli. In base ai rischi relativi al prodotto e al progetto, e al modello del ciclo di vita dello sviluppo software adottato, diverse persone possono assumere il ruolo di tester nei diversi livelli di test. Ad esempio, a livello di test di componente e di test di integrazione dei componenti, il ruolo di tester è spesso svolto dagli sviluppatori. A livello di test di accettazione, il ruolo di tester è spesso svolto da Business Analyst, esperti del dominio applicativo e utenti. A livello di test di sistema e di test di integrazione dei sistemi, il ruolo di tester viene spesso svolto da un team di test indipendente. Al livello di Operational Acceptance Test, il ruolo di tester viene spesso svolto dallo staff di Operations (esercizio) e/o di system administration.

5.2 Pianificazione e Stima dei Test

5.2.1 Scopo e Contenuto di un Test Plan

Un Test Plan delinea le attività di test per i progetti di sviluppo e di manutenzione. La pianificazione è influenzata dalla politica e dalla strategia di test dell'organizzazione, dai cicli di vita e dalle metodologie di sviluppo utilizzati (si veda il paragrafo 2.1), dall'ambito del test, dagli obiettivi, dai rischi, dai vincoli, dalle criticità, dalla testabilità e dalla disponibilità delle risorse.

Man mano che le pianificazioni di progetto e dei test procedono, diventano disponibili maggiori informazioni, e maggiori dettagli possono essere inclusi nel Test Plan. La pianificazione dei test è un'attività continuativa, che viene eseguita durante tutto il ciclo di vita del prodotto. (Si noti che il ciclo di vita del prodotto può estendersi oltre l'ambito di un progetto per includere la fase di manutenzione). I feedback dalle attività di test dovrebbero essere utilizzati per riconoscere modifiche ai rischi, in modo che la pianificazione possa essere adattata. La pianificazione può essere documentata in un Master Test Plan e in Test Plan separati per i diversi livelli di test, come il testing di sistema e il testing di accettazione, o separati per tipi di test specifici, come il testing di usabilità e il performance testing. Le attività di pianificazione dei test possono includere le seguenti attività, alcune delle quali possono essere documentate in un Test Plan:

- Determinare l'ambito, gli obiettivi e i rischi del test
- Definire l'approccio generale del test
- Integrare e coordinare le attività di test nelle attività del ciclo di vita del software
- Prendere decisioni su cosa testare, sulle persone e altre risorse richieste per svolgere le varie attività di test, e su come tali attività di test verranno portate avanti
- Schedulare le attività di analisi, progettazione, implementazione, esecuzione e valutazione dei test, sia in momenti particolari (ad es. nello sviluppo sequenziale) sia nel contesto di ciascuna iterazione (ad es. nello sviluppo iterativo)
- Selezionare le metriche per il monitoraggio e il controllo dei test
- Inserire le attività di test nel budget
- Determinare il livello di dettaglio e la struttura della documentazione di test (ad es. fornendo template o documenti di esempio)

Il contenuto dei Test Plan varia e può estendersi rispetto agli argomenti sopra identificati. Un esempio di struttura del Test Plan e un esempio di Test Plan possono essere trovati nello standard ISO (ISO/IEC/IEEE 29119-3).

5.2.2 Strategia di Test e Approccio di Test

Una strategia di test fornisce una descrizione generalizzata del processo di test, solitamente a livello di prodotto o a livello organizzativo. I tipi comuni di strategie di test includono:

- **Analitica**: questo tipo di strategia di test è basata sull'analisi di alcuni fattori (ad es. requisito o rischio). Il testing basato sul rischio è un esempio di approccio analitico, in cui i test sono progettati e prioritizzati in base al livello di rischio.
- **Model-based**: in questo tipo di strategia di test, i test sono progettati sulla base dei modelli di alcuni aspetti richiesti del prodotto, come una funzione, un processo di business, una struttura interna o una caratteristica non-funzionale (ad es. affidabilità). Esempi di tali modelli includono: modelli di processi di business, modelli degli stati e modelli di crescita dell'affidabilità.
- **Metodica**: questo tipo di strategia di test si basa sull'utilizzo sistematico di un insieme predefinito di test o di condizioni di test, come una tassonomia di tipi di failure comuni o probabili, una lista di caratteristiche di qualità importanti, oppure standard aziendali di look-and-feel per le app mobile o le pagine web.
- **Conforme al processo** (o **conforme a standard**): questo tipo di strategia di test prevede l'analisi, la progettazione e l’implementazione dei test sulla base di regole e standard esterni, come standard industriali specifici, documentazione di processo, identificazione e uso rigorosi della base di test, oppure qualsiasi processo o standard imposto alla/ dalla organizzazione.
- **Diretta** (o **consultiva**): questo tipo di strategia di test è guidata principalmente da consigli, suggerimenti o istruzioni di stakeholder, di esperti di dominio di business, o esperti di tecnologia, che possono essere esterni al team di test o aldifuori dell'organizzazione stessa.
- **Avversa alla regressione**: questo tipo di strategia di test è motivata dal desiderio di evitare la regressione delle funzionalità esistenti. Questa strategia di test include il riutilizzo del testware esistente (in particolare test case e dati di test), un’ampia automazione dei regression test e l’uso di test suite standard.
- **Reattiva**: in questo tipo di strategia di test, piuttosto che svolgere una pianificazione prima del testing (come accade nelle strategie precedenti), il testing è reattivo al componente o sistema sotto test e agli eventi che si verificano durante l'esecuzione dei test,. I test sono progettati e implementati, e possono essere immediatamente eseguiti sulla base delle conoscenze

acquisite dai risultati dei test precedenti. Il testing esplorativo è una tecnica comunemente utilizzata nelle strategie reattive.

Una strategia di test appropriata viene spesso creata combinando diversi tipi di queste strategie di test. Ad esempio, il testing basato sul rischio (strategia analitica) può essere combinato con il testing esplorativo (strategia reattiva); queste strategie si complementano a vicenda e, se usate insieme, possono raggiungere un testing più efficace.

Mentre la strategia di test fornisce una descrizione generalizzata del processo di test, l'approccio di test adatta la strategia a un particolare progetto o rilascio. L'approccio di test è il punto di partenza per selezionare tecniche di test, livelli di test e tipi di test, e per definire criteri di ingresso e criteri di uscita (rispettivamente “definition of ready” e “definition of done”). Il tailoring della strategia si basa su decisioni prese in relazione alla complessità e agli obiettivi di progetto, al tipo di prodotto che deve essere sviluppato e all’analisi dei rischi di prodotto. L'approccio selezionato dipende dal contesto e può prendere in considerazione diversi fattori, come rischi, safety, risorse disponibili e competenze, tecnologia, natura del sistema (ad es. sistema customizzato rispetto a un sistema COTS), obiettivi del test e normative.

5.2.3 Criteri di Ingresso e Criteri di Uscita (“Definition of Ready” e “Definition of Done”)

Per esercitare un controllo efficace sulla qualità del software e del testing, è consigliabile avere dei criteri che definiscono quando una specifica attività di test dovrebbe iniziare e quando tale attività può essere considerata completa. I criteri di ingresso (o “definition of ready” nello sviluppo Agile) definiscono le precondizioni per iniziare una determinata attività di test. Se i criteri di ingresso non sono soddisfatti, è probabile che l'attività si dimostrerà più difficile, più dispendiosa in termini di tempo, più costosa e più rischiosa. I criteri di uscita (o “definition of done” nello sviluppo Agile) definiscono quali condizioni devono essere raggiunte per dichiarare completato un livello di test o un insieme di test. I criteri di ingresso e uscita dovrebbero essere definiti per ciascun livello di test e tipo di test, e differiranno in base agli obiettivi del test.

Tipici criteri di ingresso includono:

- Disponibilità di requisiti, user story e/o modelli (ad es. quando si segue una strategia di test model-based) testabili
- Disponibilità di elementi di test che soddisfano i criteri di uscita di ogni livello precedente di test
- Disponibilità dell'ambiente di test
- Disponibilità degli strumenti di test necessari
- Disponibilità di dati di test e di altre risorse necessarie

Tipici criteri di uscita includono:

- I test pianificati sono stati eseguiti
- Un livello definito di copertura (ad es. di requisiti, user story, criteri di accettazione, rischi, codice) è stato raggiunto
- Il numero di difetti irrisolti è compreso entro un limite concordato
- Il numero stimato di difetti rimanenti è sufficientemente basso
- I livelli raggiunti di affidabilità, efficienza delle prestazioni, usabilità, sicurezza e di altre importanti caratteristiche di qualità sono considerati sufficienti

Anche se i criteri di uscita non sono soddisfatti, è possibile che le attività di test vengano ridotte a causa di indisponibilità di budget, del completamento dei tempi schedulati e/o di pressioni per rilasciare il prodotto sul mercato. In tali circostanze può essere accettabile terminare comunque il testing, se gli stakeholder di progetto e i responsabili di business hanno verificato e accettato il rischio di andare in produzione senza ulteriore testing.

5.2.4 Schedulazione di Esecuzione dei Test

Una volta che i test case e le procedure di test sono stati prodotti (con alcune procedure di test potenzialmente automatizzate) e organizzati in test suite, le test suite possono essere organizzate in una schedulazione di esecuzione dei test, che definisce l'ordine in cui tali test devono essere eseguiti. La schedulazione di esecuzione dei test dovrebbe tener conto di fattori come la prioritizzazione, le dipendenze, i test confermativi, i regression test e la sequenza più efficiente per l’esecuzione dei test.

Idealmente i test case dovrebbero essere ordinati in base ai loro livelli di priorità, normalmente eseguendo prima i test case con la priorità più alta. Tuttavia questa pratica può non funzionare se i test case o le funzionalità da testare hanno dipendenze. Se un test case con una priorità più alta è dipendente da un test case con una priorità più bassa, il test case con priorità più bassa deve essere eseguito prima. Allo stesso modo, se esistono dipendenze tra i test case, devono essere ordinati in modo appropriato indipendentemente dalle loro priorità relative. Anche i test confermativi e i regression test devono essere prioritizzati in base all’importanza di avere un rapido feedback sulle modifiche, ma anche in questo caso possono essere applicate delle dipendenze.

In alcuni casi, sono possibili diverse sequenze di test, ognuna delle quali con un diverso livello di efficienza. In questi casi, devono essere accettati compromessi tra l'efficienza dell'esecuzione dei test e l’aderenza alla prioritizzazione.

5.2.5 Fattori che Influenzano l’Effort del Test

La stima dell’effort del test considera la previsione della quantità di lavoro correlato ai test che sarà richiesto per soddisfare gli obiettivi del testing per un particolare progetto, rilascio o iterazione. I fattori che influenzano l’effort del test includono le caratteristiche del prodotto, le caratteristiche del processo di sviluppo, le caratteristiche delle persone e i risultati del test, come mostrato di seguito.

Caratteristiche del prodotto

- Rischi associati al prodotto
- Qualità della base di test
- Dimensione del prodotto
- Complessità del dominio di prodotto
- Requisiti per le caratteristiche di qualità (ad es. sicurezza, affidabilità)
- Livello di dettaglio richiesto per la documentazione di test
- Requisiti di conformità legale e normativa

Caratteristiche del processo di sviluppo

- Stabilità e maturità dell'organizzazione
- Modello di sviluppo in uso
- Approccio di test
- Strumenti utilizzati
- Processo di test
- Vincoli temporali

Caratteristiche delle persone

- Competenza ed esperienza delle persone coinvolte, in particolare in progetti e prodotti similari (ad es. conoscenza del dominio)
- Coesione e leadership del team

Risultati dei test

- Numero e severità dei difetti rilevati
- Quantità di rework richiesto

5.2.6 Tecniche di Stima del Test

Esistono numerose tecniche di stima utilizzate per determinare l’effort richiesto per svolgere un testing adeguato. Due delle tecniche più comunemente utilizzate sono:

- Tecnica metrics-based: l’effort del test viene stimato sulla base di metriche di precedenti progetti similari, o basandosi su valori tipici
- Tecnica expert-based: l’effort del test viene stimato sulla base dell'esperienza dei responsabili delle attività di test o di esperti

Ad esempio, nello sviluppo Agile, i grafici burndown chart sono esempi di approccio metrics-based, poiché l’effort rimanente viene valutato, e documentato, e viene quindi utilizzato, insieme alla velocità del team, per stimare la quantità di lavoro che il team può svolgere nella successiva iterazione. Il “planning poker” invece è un esempio di approccio expert-based, poichè i membri del team stimano l’effort necessario per rilasciare una funzionalità basandosi sulla propria esperienza (si veda ISTQB-CTFL-AT).

Nel modello di sviluppo sequenziale, i modelli di rimozione dei difetti sono esempi di approccio metrics-based: i volumi dei difetti e il tempo necessario per eliminarli sono valutati e documentati, per poter poi fornire una base di stima per progetti futuri di natura similare. La tecnica di stima Wideband Delphi è un esempio di approccio expert-based, in cui un gruppo di esperti fornisce stime basate sulla propria esperienza (si veda ISTQB-CTAL-TM).

5.3 Monitoraggio e Controllo dei Test

Lo scopo del monitoraggio dei test è raccogliere informazioni e fornire feedback e visibilità sulle attività di test. Le informazioni da monitorare possono essere raccolte manualmente o automaticamente e dovrebbero essere utilizzate per valutare l’avanzamento dei test e per misurare se i criteri di uscita del test (o le attività di test associate a una “definition of done” di un progetto Agile) sono soddisfatti, come il raggiungimento degli obiettivi per la copertura dei rischi di prodotto, dei requisiti o dei criteri di accettazione.

Il controllo dei test descrive le linee guida o le azioni correttive adottate a seguito di informazioni e metriche raccolte e (possibilmente) documentate. Le azioni possono coprire qualsiasi attività di test e possono influenzare qualsiasi attività del ciclo di vita del software.

Esempi di azioni di controllo dei test includono:

- Ri-prioritizzare i test quando si verifica un rischio identificato (ad es. software consegnato in ritardo)
- Modificare la schedulazione dei test, a causa della disponibilità o indisponibilità di un ambiente di test o di altre risorse
- Rivalutare se un elemento di test soddisfa un criterio di ingresso o di uscita, a causa di un rework

5.3.1 Metriche Usate nel Testing

Le metriche possono essere raccolte durante e alla fine delle attività di test, al fine di valutare:

- L’avanzamento rispetto alla schedulazione e al budget pianificati
- La qualità attuale dell'oggetto di test
- L’adeguatezza dell'approccio di test

- L’efficacia delle attività di test rispetto agli obiettivi

Comuni metriche di test includono:

- Percentuale del lavoro svolto nella preparazione dei test case rispetto al pianificato (o percentuale dei test case implementati rispetto ai pianificati)
- Percentuale del lavoro svolto nella preparazione dell'ambiente di test rispetto al pianificato
- Esecuzione dei test case (ad es. numero di test case eseguiti/non eseguiti, test case passati/falliti, e/o condizioni di test passate/fallite)
- Informazioni sui difetti (ad es. densità dei difetti, difetti rilevati e corretti, tasso di failure e risultati dei test confermativi)
- Copertura del test in termini di requisiti, user story, criteri di accettazione, rischi o codice
- Completamento delle attività, allocazione e utilizzo delle risorse, e relativo effort
- Costo del testing, che include il costo rispetto al beneficio di tilevare il successivo difetto o il costo rispetto al beneficio di eseguire il test successivo

5.3.2 Scopo, Contenuto e Destinatari dei Test Report

Lo scopo del reporting dei test è di riassumere e comunicare le informazioni sull'attività di test, sia durante che alla fine di un'attività di test (ad es. un livello di test). Il test report preparato durante un'attività di test può essere definito come Test Progress Report; un report preparato alla fine di un'attività di test può essere definito come Test Summary Report.

Durante il monitoraggio e il controllo dei test, il Test Manager prepara regolarmente i Test Progress Report per gli stakeholder. In aggiunta al contenuto comune dei Test Progress Report e Test Summary Report, tipici Test Progress Report possono includere anche:

- Lo stato delle attività di test e l’avanzamento rispetto al Test Plan
- I fattori che ne impediscono l’avanzamento
- I test pianificati per il prossimo periodo di riferimento
- La qualità dell'oggetto di test

Quando vengono raggiunti i criteri di uscita, il Test Manager prapara il Test Summary Report. Questo report fornisce un riepilogo del testing eseguito, sulla base dell'ultimo Test Progress Report e di ogni altra informazione pertinente.

Tipici Test Summary Report possono includere:

- Riepilogo dei test eseguiti
- Informazioni su quello che è accaduto durante un periodo di test
- Deviazioni dal piano, comprese deviazioni nella schedulazione, nella durata o nell’effort delle attività di test
- Stato del testing e della qualità del prodotto rispetto ai criteri di uscita (o “definition of done”)
- Fattori che hanno bloccato o continuano a bloccare l’avanzamento
- Metriche relative ai difetti, ai test case, alla copertura dei test, all’avanzamento dell'attività e all’utilizzo delle risorse (come descritto al paragrafo 5.3.1)
- Rischi residui (si veda il paragrafo 5.5)
- Prodotti di lavoro del test che possono essere riutilizzabili

Il contenuto di un test report varia in base al progetto, a requisiti organizzativi e al ciclo di vita dello sviluppo software. Ad esempio, un progetto complesso con molti stakeholder o un progetto normativo

possono richiedere un reporting più dettagliato e rigoroso, rispetto al reporting di un semplice aggiornamento software. Come altro esempio, nello sviluppo Agile, i Test Progress Report possono essere incorporati nelle task board, nei summary sui difetti e nei burndown charts, e possono essere discussi durante il daily stand-up meeting (si veda ISTQB-CTFL-AT).

In aggiunta al tailoring (adattamento) dei test report in base al contesto del progetto, i test report possono essere adattati in base ai loro destinatari. Il tipo e la quantità di informazioni che dovrebbero essere inclusi per un audience tecnica o un team di test possono essere differenti da quelli inclusi in un Executive Summary Report. Nel primo caso, può essere importante riportare informazioni dettagliate sui tipi di difetti e sul relativo andamento. Nel secondo caso, può essere più appropriato un report di alto livello (ad es. un riepilogo sullo stato dei difetti suddivisi per priorità, budget, schedulazione, e sulle condizioni di test passate/fallite/non testate).

Lo standard ISO (ISO/IEC/IEEE 29119-3) fa riferimento a due tipi di test report, il Test Progress Report e il Test Completion Report (chiamato Test Summary Report in questo Syllabus) e riporta strutture ed esempi per ogni tipo.

5.4 Configuration Management (Gestione della Configurazione)

Lo scopo del Configuration Management è stabilire e mantenere l'integrità del componente o sistema, del testware e delle loro relazioni reciproche durante il ciclo di vita del progetto e del prodotto.

Per supportare in modo appropriato il testing, il Configuration Management può dover garantire quanto segue:

- Tutti gli elementi di test sono identificati in modo univoco, sono sottoposti a controllo della versione, sono tracciate le modifiche e sono correlati tra loro
- Tutti gli elementi del testware sono identificati in modo univoco, sono sottoposti a controllo di versione, sono tracciate le modifiche, sono correlati tra loro e alle versioni degli elementi di test, in modo da poter mantenere la tracciabilità durante tutto il processo di test
- Tutti i documenti e gli elementi del software sono identificati in modo univoco nella documentazione di test

Durante la pianificazione dei test, dovrebbero essere identificate e implementate le procedure e l'infrastruttura (strumenti) di Configuration Management.

5.5 Rischi e Testing

5.5.1 Definizione di Rischio

Il rischio consiste nella possibilità che un evento abbia in futuro conseguenze negative. Il livello di rischio è determinato dalla probabilità dell'evento e dall’impatto (il danno) dell’evento.

5.5.2 Rischi di Prodotto e di Progetto

Il rischio di prodotto consiste nella possibilità che un prodotto di lavoro (ad es. una specifica, un componente, un sistema o un test) possa non soddisfare le esigenze legittime dei suoi utenti e/o degli stakeholder. Quando i rischi di prodotto sono associati a specifiche caratteristiche di qualità di un prodotto (ad es. adeguatezza funzionale, affidabilità, efficienza delle prestazioni, usabilità, sicurezza, compatibilità, manutenibilità e portabilità), sono chiamati anche rischi di qualità. Esempi di rischi di prodotto includono:

- Il software potrebbe non eseguire le funzioni previste in accordo con le specifiche
- Il software potrebbe non eseguire le funzioni richieste in accordo con le esigenze dell'utente, del cliente e/o degli stakeholder

- Un'architettura di sistema può non supportare adeguatamente alcuni requisiti non- funzionali
- Un calcolo particolare può essere eseguito, in alcune circostanze, in modo non corretto
- Una struttura di controllo di un ciclo potrebbe essere codificata in modo errato
- I tempi di risposta possono non essere adeguati per un sistema di elaborazione delle transazioni ad alte prestazioni
- Il feedback sulla User Experience (UX) potrebbe non soddisfare le aspettative del prodotto

Il rischio di progetto consiste in situazioni che, qualora dovessero verificarsi, potrebbero avere un effetto negativo sulla capacità del progetto di raggiungere i suoi obiettivi. Esempi di rischi di progetto includono:

- Problemi di progetto:
  - Possono verificarsi ritardi nel rilascio, nel completamento delle attività o nella soddisfazione dei criteri di uscita (o “definition of done”)
  - Stime non accurate, riallocazione di fondi a progetti più prioritari o taglio generale ai costi in tutta l'organizzazione possono portare a finanziamenti inadeguati
  - Modifiche tardive aumentano la possibilità di significativi rework
- Problemi organizzativi:
  - Competenze, formazione e personale possono non essere sufficienti
  - Problemi del personale possono causare conflitti e problemi
  - Gli utenti, lo staff di business o esperti in materia possono non essere disponibili a causa di priorità di business in conflitto
- Problemi politici:
  - I tester possono non comunicare le loro esigenze e/o i risultati dei test in modo adeguato
  - Gli sviluppatori e/o i tester possono non dar seguito alle informazioni rilevate durante il testing e le review (ad es. non migliorando le pratiche di sviluppo e test)
  - Può esistere un atteggiamento o un’aspettativa non appropriata verso il testing (ad es. non viene apprezzato il valore di rilevare difetti durante il testing)
- Problemi tecnici:
  - I requisiti possono non essere definiti sufficientemente bene
  - I requisiti possono non essere soddisfatti, a causa di vincoli esistenti
  - L'ambiente di test può non essere pronto nei tempi previsti
  - La pianificazione della conversione dei dati, della migrazione dei dati e il relativo supporto di strumenti può essere in ritardo
  - I punti deboli nel processo di sviluppo possono impattare la consistenza o la qualità dei prodotti di lavoro del progetto, come la progettazione, il codice, la configurazione, i dati di test e i test case
  - Un defect management scarso e problemi similari possono causare un accumulo di difetti e altri problemi tecnici
- Problemi del fornitore:
  - Una terza parte può non riuscire a rilasciare un prodotto/servizio necessario, oppure può andare in bancarotta
  - Problemi contrattuali possono causare problemi al progetto

I rischi di progetto possono influenzare sia le attività di sviluppo che le attività di test. In alcuni casi, i Project Manager hanno la responsabiolità di gestire tutti i rischi di progetto, ma non è insolito che i Test Manager abbiano la responsabilità dei rischi di progetto correlati al testing.

5.5.3 Testing basato sul Rischio e Qualità del Prodotto

Il rischio viene usato per focalizzare l’effort richiesto durante il testing. E’ usato per decidere dove e quando iniziare il testing e per identificare le aree che richiedono maggiore attenzione. Il testing viene utilizzato per ridurre la probabilità del verificarsi di un evento negativo, o per ridurre l'impatto di un evento negativo. Il testing è usato come attività di mitigazione del rischio, per fornire informazioni sui rischi identificati e sui rischi residui (irrisolti).

Un approccio di test basato sul rischio offre valide opportunità per ridurre i livelli del rischio di prodotto. Consiste nell'analisi del rischio di prodotto, che comprende l'identificazione dei rischi di prodotto e la valutazione della probabilità e dell'impatto di ciascun rischio. Le informazioni risultanti sul rischio di prodotto vengono utilizzate per guidare la pianificazione dei test, la specifica, la preparazione e l’esecuzione dei test case, il monitoraggio e il controllo dei test. Analizzare presto i rischi di prodotto contribuisce al successo di un progetto.

In un approccio basato sul rischio, i risultati dell'analisi del rischio di prodotto sono utilizzati per:

- Determinare le tecniche di test da utilizzare
- Determinare i livelli e i tipi di test specifici che devono essere eseguiti (ad es. test di sicurezza, test di accessibilità)
- Determinare l'estensione del testing da effettuare
- Prioritizzare il testing nel tentativo di individuare i difetti critici il prima possibile
- Determinare se potrebbero essere svolte attività aggiuntive rispetto al testing per ridurre il rischio (ad es. fornendo una formazione adeguata a progettisti inesperti)

Il testing basato sul rischio si basa sulla conoscenza collettiva e sulle intuizioni degli stakeholder di progetto per condurre l’analisi dei rischi di prodotto. Al fine di garantire che sia minimizzata la probabilità di un failure di prodotto, le attività di Risk Management (gestione del rischio) forniscono un approccio disciplinato per:

- Analizzare (e rivalutare su base regolare) cosa potrebbe non funzionare (rischi)
- Determinare quali sono i rischi importanti da affrontare
- Implementare azioni per mitigare questi rischi
- Predisporre contingency plan (piani di contingenza) per affrontare i rischi nel caso in cui diventino eventi reali

Inoltre, il testing può identificare nuovi rischi, aiutare a determinare quali rischi dovrebbero essere mitigati e ridurre l’incertezza relativa ai rischi.

5.6 Defect Management (Gestione dei Difetti)

Uno degli obiettivi del test è trovare difetti, quindi i difetti rilevati durante il testing dovrebbero essere tracciati. Il modo in cui i difetti sono tracciati può variare, in base al contesto del componente o sistema da testare, al livello di test e al modello del ciclo di vita dello sviluppo software. I difetti identificati dovrebbero essere investigati e tracciati dalla loro scoperta e classificazione fino alla loro risoluzione (ad es. correzione dei difetti e testing confermativo con esito positivo, posticipo a una release successiva, accettazione intesa come limitazione permanente del prodotto, ecc.). Al fine di gestire tutti i difetti fino alla loro risoluzione, un organizzazione dovrebbe stabilire un processo di Defect Management che includa un workflow e delle regole per la classificazione. Questo processo deve essere concordato con tutti coloro che partecipano al processo di Defect Management, inclusi architetti,

progettisti, sviluppatori, tester e Product Owner. In alcune organizzazioni il logging e il tracciamento dei difetti possono essere molto informali.

Durante il processo di Defect Management, alcuni report possono descrivere falsi positivi, cioè viene segnalato un failure nonostante non esista nessun difetto nell’oggetto del test. Ad esempio, un test può fallire quando una connessione di rete viene interrotta o va in time-out. Questo comportamento non deriva da un difetto nell'oggetto di test, ma è un'anomalia che richiede di essere investigata. I tester dovrebbero cercare di minimizzare il numero di falsi positivi segnalati come difetti.

I difetti possono essere tracciati durante la codifica, l'analisi statica, le review, o durante il testing dinamico, oppure durante l'utilizzo di un prodotto software. I difetti possono essere segnalati per problemi nel codice, in sistemi in utilizzo o in qualsiasi tipo di documentazione, inclusi requisiti, user story e criteri di accettazione, documenti di sviluppo, documenti di test, manuali utente o guide all'installazione. Al fine di avere un processo di Defect Management che sia efficace ed efficiente, le organizzazioni possono definire degli standard per gli attributi, la classificazione e il workflow dei difetti.

Tipici defect report hanno i seguenti obiettivi:

- Fornire a sviluppatori e altri stakeholder informazioni su eventuali eventi negativi che si sono verificati, per permettere loro di identificarne gli effetti specifici, di isolare il problema con un test minimale che lo riproduca, e di correggere il potenziale difetto, se necessario, o ppure di risolvere il problema
- Fornire ai Test Manager un mezzo per tenere traccia della qualità del prodotto di lavoro e dell'impatto sul testing (ad es. se vengono segnalati molti difetti, i tester avranno dedicato molto tempo a tracciarli, invece di eseguire altri test e sarà necessario eseguire più testing confermativo)
- Fornire suggerimenti per il miglioramento del processo di sviluppo e di test

Un defect report creato durante il testing dinamico include in genere:

- Un identificativo
- Un titolo e un breve riassunto del difetto segnalato
- La data del defect report, l’organizzazione e l’autore che ha tracciato il difetto
- L’identificativo dell'elemento di test (configuration item sotto test) e l'ambiente
- La fase del ciclo di vita dello sviluppo in cui è stato osservato il difetto
- Una descrizione del difetto per consentirne la riproduzione e la risoluzione, compresi log, dump del database, screenshot o registrazioni (se rilevato durante l'esecuzione dei test)
- Risultato atteso e risultato effettivo
- Ambito o grado di impatto (severità) del difetto sugliinteressi degli stakeholder
- Urgenza/priorità della correzione
- Stato del defect report (ad es. aperto, differito, duplicato, in attesa di risoluzione, in attesa di testing confermativo, riaperto, chiuso)
- Conclusioni, raccomandazioni e approvazioni
- Argomenti generici, come l’indicazione di altre aree che possono essere interessate dalle modifiche derivanti dal difetto
- Storia delle modifiche, come la sequenza di azioni svolte dai membri del team di progetto per isolare, correggere e confermarne la risoluzione del difetto
- Riferimenti, incluso il test case che ha rivelato il problema.

Se si utilizzano strumenti di Defect Management, alcuni di questi dettagli possono essere inclusi e/o gestiti automaticamente, ad esempio l'assegnazione automatica di un identificativo, l'assegnazione e l'aggiornamento dello stato del defect report nel workflow, ecc.. I difetti rilevati durante il testing statico,

in particolare nelle review, sono normalmente documentati in modo diverso, ad es. nelle minute del review meeting.

Un esempio del contenuto di un defect report è presente nello standard ISO (ISO/IEC/IEEE 29119-3) (che tratta i defect report come incident report).

6. Strumenti a Supporto del Testing – 40 minuti

Parole Chiave

strumenti di esecuzione dei test, strumenti di Test Management, test automation, testing data-driven, testing keyword-driven

Obiettivi di Apprendimento per gli Strumenti a Supporto del Testing

6.1 Considerazioni sugli Strumenti di Test

FL-6.1.1 (K2) Classificare gli strumenti di test in base al loro scopo e alle attività di test che supportano

FL-6.1.2 (K1) Identificare benefici e rischi del test automation

FL-6.1.3 (K1) Ricordare considerazioni speciali sugli strumenti di esecuzione dei test e di Test Management

6.2 Utilizzo Efficace degli Strumenti

FL-6.2.1 (K1) Identificare i principi più importanti per la selezione di uno strumento

FL-6.2.2 (K1) Richiamare gli obiettivi nell’utilizzo di progetti pilota per introdurre strumenti

FL-6.2.3 (K1) Identificare i fattori di successo per la valutazione, l'implementazione, la distribuzione e il supporto continuativo di strumenti di test in un'organizzazione

6.1 Considerazioni sugli Strumenti di Test

Gli strumenti di test possono essere utilizzati per supportare una o più attività di test. Tali strumenti includono:

- Strumenti direttamente utilizzati nel testing, come strumenti di esecuzione dei test e strumenti di preparazione dei dati di test
- Strumenti che aiutano a gestire requisiti, test case, procedure di test, test script automatizzati, risultati dei test, dati di test e difetti, e che permettono di eseguire il reporting e il monitoraggio dell’esecuzione dei test
- Strumenti utilizzati per l’analisi e la valutazione
- Qualsiasi strumento che supporti il testing (in tal senso, un foglio di calcolo è da considerare uno strumento di test)

6.1.1 Classificazione degli Strumenti di Test

Gli strumenti di test possono avere uno o più dei seguenti scopi, in base al contesto:

- Migliorare l'efficienza delle attività di test automatizzando attività ripetitive o attività che richiedono risorse significative se eseguite manualmente (ad es. esecuzione dei test, regression testing)
- Migliorare l'efficienza delle attività di test supportando le attività di test manuali durante tutto il processo (si veda il paragrafo 1.4)
- Migliorare la qualità delle attività di test attraverso test più consistenti e un livello più elevato di riproducibilità del difetto
- Automatizzare attività che non possono essere eseguite manualmente (ad es. performance testing su larga scala)
- Aumentare l'affidabilità del testing (ad es. automatizzando il confronto di grandi quantità di dati o simulando un comportamento)

Gli strumenti possono essere classificati in base a diversi criteri, come lo scopo, il costo, il modello di licenza (ad es. commerciale oppure open source) e la tecnologia utilizzata. Gli strumenti sono classificati in questo Syllabus in base alle attività di test supportate.

Alcuni strumenti supportano solo una o principalmente un'attività; altri possono supportare più di un’attività, ma sono classificati rispetto all'attività che supporta maggiormente. Gli strumenti di un unico fornitore, specialmente quelli che sono stati progettati per funzionare insieme, possono essere forniti come suite integrata.

Alcuni tipi di strumenti di test possono essere intrusivi, il che significa che possono influenzare il risultato effettivo del test. Ad esempio, i tempi di risposta effettivi di un'applicazione potrebbero essere differenti a causa di istruzioni aggiuntive che vengono eseguite da uno strumento di performance testing, oppure il livello di copertura del codice ottenuto potrebbe essere distorto a causa dell'utilizzo di uno strumento di copertura. La conseguenza dell'uso di strumenti intrusivi è chiamata effetto sonda (“probe effect”).

Alcuni strumenti offrono un tipo di supporto generalmente più appropriato per gli sviluppatori (ad es. strumenti che vengono utilizzati durante il testing di componente e di integrazione). Tali strumenti sono contrassegnati con "(D)" nei paragrafi seguenti.

Strumenti a supporto della gestione del testing e del testware

Gli strumenti di gestione possono essere applicati a qualsiasi attività di test durante l'intero ciclo di vita dello sviluppo software. Esempi di strumenti che supportano il Test Management e il testware includono:

- Strumenti di Test Management e strumenti di Application Lifecycle Management (ALM, gestione del ciclo di vita dell’applicativo)
- Strumenti di Requirement Management (gestione dei requisiti) (ad es. per la tracciabilità degli oggetti di test)
- Strumenti di Defect Management
- Strumenti di Configuration Management
- Strumenti di Continuous Integration (D)

Strumenti a supporto del testing statico

Gli strumenti per il testing statico sono associati alle attività e ai benefici descritti nel capitolo 3. Esempi di tali strumenti includono:

- Strumenti di analisi statica (D)

Strumenti a supporto della progettazione e implementazione dei test

Gli strumenti di progettazione dei test aiutano nella creazione di prodotti di lavoro manutenibili in fase di progettazione e implementazione dei test, che includono test case, procedure di test e dati di test. Esempi di tali strumenti includono:

- Strumenti di testing model-based
- Strumenti di preparazione dei dati di test

In alcuni casi, gli strumenti che supportano la progettazione e l'implementazione dei test possono anche supportare l'esecuzione e il logging dei test, oppure fornire i loro output direttamente verso altri strumenti che supportano l'esecuzione e il logging dei test.

Strumenti a supporto dell’esecuzione e del logging dei test

Esistono molti strumenti per supportare e migliorare le attività di esecuzione e logging dei test. Esempi di questi strumenti includono:

- Strumenti di esecuzione dei test (ad es. per eseguire i regression test)
- Strumenti di copertura (ad es. copertura dei requisiti, copertura del codice (D))
- Strumenti di test harness (D)

Strumenti a supporto della misura delle prestazioni e dell’analisi dinamica

Gli strumenti per la misura delle prestazioni e gli strumenti di analisi dinamica sono essenziali per supportare attività di performance testing e di test di carico, poichè tali attività non possono essere effettivamente svolte manualmente. Esempi di questi strumenti includono:

- Strumenti di performance testing
- Strumenti di analisi dinamica (D)

Strumenti a supporto di esigenze di testing specializzato

In aggiunta agli strumenti che supportano il processo generale di test, esistono molti altri strumenti che supportano attività di testing più specifiche per le caratteristiche non-funzionali.

6.1.2 Benefici e Rischi del Test Automation

La semplice acquisizione di uno strumento non garantisce il suo successo. Ogni nuovo strumento introdotto in un'organizzazione richiederà un effort per ottenere benefici reali e duraturi. Esistono potenziali benefici e opportunità nell’uso di strumenti nel testing, ma esistono anche rischi. Questo è

particolarmente vero per gli strumenti di esecuzione dei test (spesso indicati come strumenti di test automation).

Potenziali benefici legati all'utilizzo di strumenti a supporto dell'esecuzione dei test includono:

- Riduzione del lavoro manuale ripetitivo (ad es. esecuzione di regression test, attività di installazione/disinstallazione dell'ambiente, re-inserimento degli stessi dati di test e verifica rispetto a standard di codifica), con conseguente risparmio di tempo
- Maggiore consistenza e ripetibilità (ad es. dati di test creati in modo coerente, test eseguiti dallo strumento nello stesso ordine e con la stessa frequenza, e test derivati dai requisiti in modo consistente)
- Valutazione più obiettiva (ad es. misure statiche, copertura)
- Accesso più facile alle informazioni sul testing (ad es. statistiche e grafici sull'avanzamento dei test, tassi di difettosità e delle prestazioni)

Potenziali rischi legati all'utilizzo di strumenti a supporto del testing includono:

- Le aspettative sullo strumento possono essere non realistiche (incluse funzionalità e facilità d'uso)
- Il tempo, i costi e l’effort per l'introduzione iniziale di uno strumento può essere sottostimato (inclusa formazione e competenze esterne)
- Il tempo e l’effort necessari per ottenere benefici significativi e continui dallo strumento possono essere sottostimati (incluse la necessità di modifiche nel processo di test e il miglioramento continuo delle modalità di utilizzo dello strumento)
- L’effort richiesto per manutenere i prodotti di lavoro di test generati dallo strumento potrebbe essere sottostimato
- La possibilità di essere troppo confidenti nello strumento (visto in sostituzione della progettazione o dell'esecuzione dei test o utilizzo di test automatizzati quando il testing manuale risulterebbe migliore)
- Il controllo del versioning dei prodotti di lavoro dei test può essere trascurato
- I rroblemi di interfaccia e interoperabilità tra strumenti critici possono essere trascurati, come gli strumenti di Requirement Management, gli strumenti di Configuration Management, gli strumenti di Defect Management e gli strumenti provenienti da più fornitori
- Il fornitore dello strumento può cessare l'attività, ritirare lo strumento o vendere lo strumento a un fornitore differente
- Il fornitore può fornire un debole supporto nell’assistenza, negli aggiornamenti e nella correzione di difetti
- Un progetto open source può essere sospeso
- Una nuova piattaforma o tecnologia può non essere supportata dallo strumento
- La responsabilità dello strumento può non essere chiara (ad es. per attività di mentoring, aggiornamento, ecc.)

6.1.3 Considerazioni speciali per gli Strumenti di Esecuzione dei Test e di Test Management

Per introdurre uno strumento in una organizzazione in modo semplice e con successo, dovrebbero essere fatte una serie di considerazioni nel momento in cui si selezionano e si integrano strumenti di esecuzione dei Test e di Test Management.

Strumenti di esecuzione dei test

Gli strumenti di esecuzione dei test eseguono gli oggetti di test utilizzando test script automatizzati. Questo tipo di strumenti spesso richiede un effort significativo per riuscire ad ottenere benefici significativi.

- Approccio di cattura dei test (capturing test): Questo approccio cattura i test registrando le azioni eseguite da un tester manuale, ma questo approccio non è scalabile rispetto alla crescita del numero di test script. Un captured script (script catturato) è una rappresentazione lineare con specifici dati e azioni come parte di ogni script. Questo tipo di script può essere instabile quando si verificano eventi imprevisti e richiedono una manutenzione continua, poichè l'interfaccia utente del sistema evolve nel tempo.
- Approccio di test data-driven (guidato dai dati): Questo approccio separa gli input dei test e i risultati attesi, normalmente in un foglio di calcolo, e utilizza un test script più generico, che è in grado di leggere i dati di input ed eseguire lo stesso test script con dati differenti.
- Approccio di test keyword-driven (guidato da parole chiave): In questo approccio, uno script generico elabora le keyword (parole chiave) che descrivono le azioni da intraprendere (chiamate anche “action word”), le quali poi attivano gli script delle keyword per elaborare i dati di test associati.

Gli approcci sopra descritti richiedono che qualcuno abbia esperienza nel linguaggio di scripting (tester, sviluppatori o specialisti nel Test Automation). Quando si utilizzano gli approcci di test data-driven o keyword-driven, i tester che non sono familiari con il linguaggio di scripting possono anche contribuire alla creazione di dati di test e/o di keyword per questi script predefiniti. Indipendentemente dalla tecnica di scripting utilizzata, i risultati attesi di ogni test necessitano di essere confrontati con i relativi risultati effettivi, in modo dinamico (mentre il test è in esecuzione) oppure tali risultati vengono memorizzati per un confronto successivo (post-esecuzione).

Ulteriori dettagli ed esempi sugli approcci di test data-driven e keyword-driven sono descrittiin ISTQB-CTAL-TAE, in Fewster 1999 e in Buwalda 2001.

Gli strumenti di Testing Model-Based (Model-Based Testing MBT) consentono di catturare una Specifica Funzionale nella forma di un modello, come un activity diagram. Questa attività viene generalmente eseguita da un progettista di sistema. Lo strumento di MBT interpreta il modello per creare specifiche di test case, che possono quindi essere salvate in uno di strumento di Test Management e/o eseguite da uno strumento di esecuzione dei test (si veda ISTQB-CTFL-MBT).

Strumenti di Test Management

Gli strumenti di Test Management hanno spesso bisogno di interfacciarsi con altri strumenti o fogli di calcolo per ragioni differenti, che includono:

- Produrre informazioni utili in un formato che si adatti alle esigenze dell'organizzazione
- Mantenere una tracciabilità consistente verso i requisiti in uno strumento di Requirement Management
- Correlare con le informazioni sulla versione dell’oggetto di test nello strumento di Configuration Management

Questo è particolarmente importante da considerare quando si utilizza uno strumento integrato (ad es. Application Lifecycle Management), che include un modulo di Test Management, ma anche altri moduli (ad es. la schedulazione di progetto e le informazioni sul budget) che vengono utilizzati da gruppi differenti all'interno di un'organizzazione.

6.2 Utilizzo Efficace degli Strumenti

6.2.1 Principi principali per la selezione di uno strumento

Le principali considerazioni nella selezione di uno strumento per un'organizzazione includono:

- Valutazione della maturità della propria organizzazione, dei suoi punti di forza e di debolezza

- Identificazione delle opportunità di miglioramento del processo di test supportato da strumenti
- Comprensione delle tecnologie utilizzate dagli oggetti di test, al fine di selezionare uno strumento che sia compatibile con tali tecnologie
- Comprensione degli strumenti di build e di continuous integration già in uso all'interno dell'organizzazione, al fine di garantire la compatibilità e l'integrazione dello strumento
- Valutazione dello strumento rispetto a requisiti chiari e criteri oggettivi
- Considerazione sulla disponibilità o meno dello strumento per un periodo free trial (e per quanto tempo)
- Valutazione del fornitore (inclusa la formazione, il supporto e gli aspetti commerciali) o del supporto per gli strumenti non commerciali (ad es. open source)
- Identificazione dei requisiti interni per il coaching e il mentoring nell'uso dello strumento
- Valutazione delle esigenze di formazione, considerando gli skill di test (e di test automation) di coloro che lavoreranno direttamente con lo strumento
- Considerazione dei pro e contro dei vari modelli di licenza (ad es. commerciale o open source)
- Stima del rapporto costi-benefici, basata su un business case concreto (se necessario)

Come passo finale, dovrebbe essere eseguita una valutazione proof-of-concept per stabilire se lo strumento funziona efficacemente con il software sotto test e all'interno dell'attuale infrastruttura o, se necessario, per identificare le modifiche necessarie a tale infrastruttura per l’utilizzo efficace dello strumento.

6.2.2 Progetti Pilota per l'Introduzione di uno Strumento in un'Organizzazione

Dopo aver completato la selezione dello strumento e un proof-of-concept di successo, l’introduzione dello strumento selezionato in un’organizzazione normalmente inizia con un progetto pilota, che ha i seguenti obiettivi:

- Acquisire una conoscenza approfondita dello strumento, che comprende sia i suoi punti di forza che i suoi punti deboli
- Valutare come lo strumento si adatta ai processi e alle pratiche esistenti, e determinare cosa sarebbe necessario modificare
- Decidere le modalità standard di utilizzo, gestione, archiviazione e manutenzione dello strumento e dei prodotti di lavoro del test (ad es. decidendo le naming convention per i file e i test, selezionando gli standard di codifica, creando le librerie e definendo la modularità delle test suite)
- Valutare se i benefici saranno raggiunti a costi ragionevoli
- Comprendere le metriche che lo strumento deve raccogliere e registrare, e configurare lo strumento per assicurare che tali metriche possano essere raccolte e documentate.

6.2.3 Fattori di Successo per gli Strumenti

I fattori di successo per la valutazione, l'implementazione, la distribuzione e il supporto continuo degli strumenti all'interno di un’organizzazione includono:

- Distribuire lo strumento al resto dell'organizzazione in modo incrementale
- Modificare e migliorare i processi per adattarli all'utilizzo dello strumento
- Fornire formazione, coaching e mentoring agli utenti dello strumento
- Definire linee guida per l'utilizzo dello strumento (ad es. standard interni per l'automazione)

- Implementare le modalità di raccolta delle informazioni dall'utilizzo effettivo dello strumento
- Monitorare l’utilizzo e i benefici dello strumento
- Fornire supporto agli utenti di uno specifico strumento
- Raccogliere “lessons learned” da tutti gli utenti

È inoltre importante assicurare che lo strumento sia integrato dal punto di vista tecnico ed organizzativo al ciclo di vita dello sviluppo software, e questo può coinvolgere organizzazioni separate, responsabili delle operations, e/o di fornitori di terze parti.

Si veda Graham 2012 per esperienze e consigli sull'utilizzo degli strumenti di esecuzione dei test.

7. Riferimenti

7.1 Standard

ISO/IEC/IEEE 29119-1 (2013) Software and systems engineering - Software testing - Part 1: Concepts and definitions

ISO/IEC/IEEE 29119-2 (2013) Software and systems engineering - Software testing - Part 2: Test processes

ISO/IEC/IEEE 29119-3 (2013) Software and systems engineering - Software testing - Part 3: Test documentation

ISO/IEC/IEEE 29119-4 (2015) Software and systems engineering - Software testing - Part 4: Test techniques

ISO/IEC 25010, (2011) Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) System and software quality models

ISO/IEC 20246: (2017) Software and systems engineering — Work product reviews

UML 2.5, Unified Modeling Language Reference Manual, http://www.omg.org/spec/UML/2.5.1/, 2017

7.2 Documenti ISTQB®

ISTQB® Glossary

ISTQB® Foundation Level Overview 2018

ISTQB-CTFL-MBT Foundation Level Model-Based Tester Extension Syllabus

ISTQB- CTFL-AT Foundation Level Agile Tester Extension Syllabus

ISTQB- CTAL-ATA Advanced Level Test Analyst Syllabus

ISTQB- CTAL-TTA Advanced Level Technical Test Analyst Syllabus

ISTQB- CTAL-ATM Advanced Level Test Manager Syllabus

ISTQB- CTAL-SEC Advanced Level Security Tester Syllabus

ISTQB- CTAL-TAE Advanced Level Test Automation Engineer Syllabus

ISTQB- CTEL-TM Expert Level Test Management Syllabus

ISTQB- CTEL-ITP Expert Level Improving the Test Process Syllabus

7.3 Libri e Articoli

Beizer, B. (1990*) Software Testing Techniques (2e)*, Van Nostrand Reinhold: Boston MA

Black, R. (2017) *Agile Testing Foundations*, BCS Learning & Development Ltd: Swindon UK

Black, R. (2009) *Managing the Testing Process (3e)*, John Wiley & Sons: New York NY

Buwalda, H. et al. (2001) *Integrated Test Design and Automation*, Addison Wesley: Reading MA

Copeland, L. (2004) *A Practitioner’s Guide to Software Test Design*, Artech House: Norwood MA

Craig, R. and Jaskiel, S. (2002) *Systematic Software Testing*, Artech House: Norwood MA

Crispin, L. and Gregory, J. (2008) *Agile Testing*, Pearson Education: Boston MA

Fewster, M. and Graham, D. (1999) *Software Test Automation*, Addison Wesley: Harlow UK

Gilb, T. and Graham, D. (1993) *Software Inspection*, Addison Wesley: Reading MA

Graham, D. and Fewster, M. (2012) *Experiences of Test Automation*, Pearson Education: Boston MA

Gregory, J. and Crispin, L. (2015) *More Agile Testing*, Pearson Education: Boston MA

Jorgensen, P. (2014) *Software Testing, A Craftsman’s Approach (4e)*, CRC Press: Boca Raton FL

Kaner, C., Bach, J. and Pettichord, B. (2002) *Lessons Learned in Software Testing*, John Wiley & Sons: New York NY

Kaner, C., Padmanabhan, S. and Hoffman, D. (2013) *The Domain Testing Workbook*, Context-Driven Press: New York NY

Kramer, A., Legeard, B. (2016) *Model-Based Testing Essentials: Guide to the ISTQB® Certified Model-Based Tester: Foundation Level*, John Wiley & Sons: New York NY

Myers, G. (2011) *The Art of Software Testing*, (3e), John Wiley & Sons: New York NY

Sauer, C. (2000) “The Effectiveness of Software Development Technical Reviews: A Behaviorally Motivated Program of Research,” *IEEE Transactions on Software Engineering*, *Volume 26, Issue 1, pp1-*

Shull, F., Rus, I., Basili, V. July 2000. “How Perspective-Based Reading can Improve Requirement Inspections.” *IEEE Computer, Volume 33, Issue 7, pp 73-79*

van Veenendaal, E. (ed.) (2004) *The Testing Practitioner* (Chapters 8 - 10), UTN Publishers: The Netherlands

Wiegers, K. (2002) *Peer Reviews in Software*, Pearson Education: Boston MA

Weinberg, G. (2008) *Perfect Software and Other Illusions about Testing*, Dorset House: New York NY

7.4 Altri Riferimenti (non direttamente referenziati in questo Syllabus)

Black, R., van Veenendaal, E. and Graham, D. (2019) *Foundations of Software Testing: ISTQB® Certification (4e)*, Cengage Learning: London UK

Hetzel, W. (1993) *Complete Guide to Software Testing (2e)*, QED Information Sciences: Wellesley MA

8. Appendice A

8.1 Storia di questo Documento

Questo documento è il Syllabus ISTQB® Certified Tester Foundation Level, il primo livello di certificazione internazionale approvato da ISTQB® (www.istqb.org).

Questo documento è stato preparato tra Giugno e Agosto 2019 da un Gruppo di Lavoro composto da membri nominati dall'International Software Testing Qualifications Board (ISTQB®). Aggiornamenti sono stati inseriti a seguito degli input della review svolta dai membri dei Board che hanno utilizzato il Syllabus Foundation 2018.

Il precedente Syllabus Foundation 2018 è stato preparato tra il 2014 e il 2018 da un gruppo di lavoro composto da membri nominati dall'International Software Testing Qualifications Board (ISTQB®). La versione 2018 è stata inizialmente sottoposta a review dai rappresentanti di tutti i Board ISTQB®, e successivamente dai professionisti provenienti dalla comunità internazionale di testing del software.

8.2 Obiettivi della Certificazione Foundation Level

- Ottenere il riconoscimento del testing come una specializzazione dell’ingegneria del software essenziale e professionale
- Fornire un framework standard per lo sviluppo delle carriere dei tester
- Consentire ai tester professionalmente qualificati di essere riconosciuti tali da datori di lavoro, clienti e colleghi, e di innalzare il profilo dei tester
- Promuovere pratiche di test valide e consistenti all'interno di tutte le discipline dell'ingegneria del software
- Identificare argomenti del testing che sono rilevanti e di valore per l’industria
- Consentire ai fornitori di software di assumere tester certificati e ottenere così un vantaggio commerciale rispetto ai loro concorrenti, pubblicizzando la loro politica di assunzione dei tester
- Fornire un'opportunità ai tester e a coloro che sono interessati al testing di acquisire una certificazione in materia riconosciuta a livello internazionale

8.3 Obiettivi della Certificazione Internazionale

- Essere in grado di confrontare la conoscenza del testing tra nazioni differenti
- Consentire ai tester di spostarsi più facilmente nelle diverse nazioni
- Consentire a progetti multinazionali/internazionali di avere una comprensione comune delle problematiche del testing
- Aumentare il numero di tester certificati in tutto il mondo
- Avere un maggiore impatto e valore come iniziativa a livello internazionale piuttosto che con un approccio specifico per ogni nazione
- Sviluppare una comune base internazionale di conoscenza e comprensione sul testing, attraverso il Syllabus e la terminologia, e aumentare il livello di conoscenza sul testing per tutti partecipanti
- Promuovere il testing come professione in più nazioni
- Consentire ai tester di ottenere una certificazione riconosciuta nella loro lingua madre
- Consentire la condivisione di conoscenze e risorse tra le nazioni

- Fornire il riconoscimento internazionale dei tester e di questa certificazione, grazie alla partecipazione di molte nazioni

8.4 Requisiti di Ingresso per la Certificazione

Il criterio di ingresso per svolgere l'esame ISTQB® Certified Tester Foundation Level è che i candidati abbiano un interesse per il testing del software. Tuttavia, è fortemente raccomandato ai candidati anche di:

- Avere almeno un background minimo nello sviluppo software o nel testing del software, come ad es. sei mesi di esperienza come tester di sistema o di user acceptance test oppure come sviluppatore software
- Partecipare a un corso che è stato accreditato da uno dei board riconosciuti da ISTQB® secondo gli standard ISTQB®.

8.5 Background e Storia della Certificazione Foundation Level nel Testing del Software

La certificazione indipendente dei tester del software è iniziata nel Regno Unito con la British Computer Society’s Information Systems Examination Board (ISEB), quando nel 1998 è stato creato un Software Testing Board (www.bcs.org.uk/iseb). Nel 2002, in Germania ASQF ha iniziato a supportare uno schema tedesco di certificazione dei tester (www.asqf.de). Questo Syllabus è basato sui Syllabi ISEB e ASQF, che sono stati riorganizzati, aggiornati e estesi nel contenuto, con un’enfasi rivolta agli argomenti che forniscono il supporto più pratico ai tester.

Una Certificazione Foundation Level nel Testing del Software (ad es. rilasciato da ISEB, ASQF o da un board riconosciuto da ISTQB®) già esistente e rilasciata prima della pubblicazione della presente Certificazione Internazionale, sarà riconosciuta equivalente alla Certificazione Internazionale. Il Certificato Foundation Level non scade e non ha bisogno di essere rinnovato. La data in cui è stata acquisita la Certificazione è indicata sul certificato.

All'interno di ogni nazione partecipante, gli aspetti locali sono controllati da un Board nazionale o regionale riconosciuto da ISTQB®. I compiti di ogni membro del Board sono specificati da ISTQB®, ma sono implementati all'interno di ogni nazione. Questi compiti attesi dei membri del Board includono l'accreditamento di Training Provider e l’organizzazione degli esami.

9. Appendice B – Obiettivi di Apprendimento/Livello Cognitivo di Conoscenza

I seguenti obiettivi di apprendimento sono definiti per essere applicati in questo Syllabus. Ogni argomento nel Syllabus sarà esaminato in accordo all’obiettivo di apprendimento associato a quell’argomento.

Livello 1: Ricordare (K1)

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

Parole chiave:

Identificare, ricordare, recuperare, richiamare alla memoria, riconoscere, conoscere.

Esempi:

Essere in grado di riconoscere la definizione di “failure” come:

- “Mancato rilascio del servizio verso un utente finale o ogni altro stakeholder” oppure
- “Deviazione del componente o sistema dal rilascio, servizio o risultato atteso”

Livello 2: Comprendere (K2)

Il candidato è in grado di selezionare le ragioni o le spiegazioni per le affermazioni legate all’argomento, ed è in grado di riassumere, confrontare, classificare, categorizzare e fornire esempi per i concetti del testing.

Parole chiave:

Riassumere, generalizzare, astrarre, classificare, comparare, mappare, esemplificare, interpretare, tradurre, rappresentare, categorizzare, inferire, concludere, costruire modelli.

Esempi:

Essere in grado di spiegare la ragione del perché l’analisi e la progettazione dei test devono essere svolte il prima possibile:

- Per trovare i difetti quando sono meno costosi da correggere
- Per trovare prima i difetti più importanti/critici

Essere in grado di spiegare le similitudini e le differenze tra il testing di integrazione e il testing di sistema:

- Similarità: sia nel testing di integrazione che nel testing di sistema gli oggetti di test includono più di un componente e possono essere inclusi tipi di test non-funzionali
- Differenze: il testing di integrazione si concentra sulle interfacce e le interazioni; il testing di sistema si concentra sugli aspetti dell’intero sistema, come l’elaborazione end-to-end

Livello 3: Applicare (K3)

Il candidato è in grado di selezionare la corretta applicazione di un concetto o di una tecnica e applicarla nell’ambito di un dato contesto.

Parole chiave:

Implementare, eseguire, utilizzare, seguire una procedura, applicare una procedura.

Esempi:

- Essere in grado di identificare i valori limite per partizioni valide ed invalide
- Essere in grado di selezionare i test case da uno state diagram per coprire tutte le transizioni

Riferimento (per i livelli cognitivi degli obiettivi di apprendimento):

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

10. Appendice C – Release Note

Il Syllabus ISTQB® Foundation 2018 V3.1 è un aggiornamento minore della release 2018. Una Release Note separata 2018 V3.1 è stata creata con una overview per singoli capitoli. In aggiunta, è stata rilasciata una versione del Syllabus ISTQB® Foundation 2018 V3.1 dove sono tracciate le modifiche.

Il Syllabus ISTQB® Foundation 2018 è un importante aggiornamento e riscrittura della versione 2011. Per questo motivo non esistono note di rilascio dettagliate per singolo capitolo e paragrafo. Tuttavia viene fornito un riassunto delle principali modifiche. In aggiunta, ISTQB® fornisce, in un documento separato di Release Note, la tracciabilità tra gli obiettivi di apprendimento del Syllabus Foundation Level versione 2011 e quelli della versione 2018, evidenziando quali sono stati aggiunti, aggiornati o eliminati.

All'inizio del 2017 più di 550.000 persone in più di 100 paesi hanno svolto l’esame Foundation e oltre 500.000 sono i tester certificati in tutto il mondo. Con l'aspettativa che tutti abbiano letto il Syllabus Foundation per essere in grado di superare l'esame, è probabile che il Syllabus Foundation sia il documento di testing del software più letto da sempre!

Perciò questo importante aggiornamento è stato fatto nel rispetto di tale eredità e per migliorare comunque il valore che ISTQB® offre alle prossime 500.000 persone della comunità globale del testing.

In questa versione tutti gli obiettivi di apprendimento sono stati modificati per renderli atomici e per creare una chiara tracciabilità tra ciascun obiettivo di apprendimento e i relativi paragrafi (e domande d'esame) e viceversa fra i paragrafi (e domande d'esame) e l'obiettivo di apprendimento associato. Inoltre, le allocazioni temporali (tempi di formazione e/o apprendimento) di ogni capitolo sono state rese più realistiche rispetto a quelle della versione 2011, utilizzando euristiche verificate e formule utilizzate in altri Syllabi ISTQB®, che si basano su un'analisi degli obiettivi di apprendimento che devono essere coperti in ogni capitolo.

Anche se questo è un Syllabus Foundation, che specifica le best practice e le tecniche che hanno mantenuto la loro validità nel tempo, sono state apportate modifiche per aggiornarne e modernizzarne il contenuto, soprattutto in termini di metodologie di sviluppo del software (ad es. Scrum e continuous deployment) e tecnologie (ad es. Internet of Things). Sono stati aggiornati gli standard di riferimento per renderli più attuali, come segue:

1. ISO/IEC/IEEE 29119 sostituisce lo standard IEEE 829.
2. ISO/IEC 25010 sostituisce ISO 9126.
3. ISO/IEC 20246 sostituisce IEEE 1028.

Inoltre, poiché il portafoglio ISTQB® è cresciuto notevolmente nell'ultimo decennio, sono stati aggiunti, dove applicabili, ampi riferimenti incrociati a materiale correlato in altri Syllabi ISTQB®, nonché è stato attentamente verificato l'allineamento a tutti i Syllabi e al Glossario ISTQB®. L'obiettivo è rendere questa versione più facile da leggere, comprendere, imparare e tradurre, focalizzandosi sull’utilità pratica e un equilibrio tra conoscenza e competenza.

Per un'analisi dettagliata delle modifiche apportate nel Syllabus Foundation 2018, consultare il ISTQB® Certified Tester Foundation Level Release Note 2018.