Certificazione di Tester

Syllabus Livello ‘Foundation’

Versione 2018

International Software Testing Qualifications Board

ITAlian Software Testing Qualifications Board

Contenuto

Storico delle Revisioni

ISTQB 2018	27-April-2018	Candidate general release version
ISTQB 2018	12-February-2018	Candidate beta version
ISTQB 2018	19-January-2018	Cross-review internal version 3.0.
ISTQB 2018	15-January-2018	Pre-cross-review internal version 2.9, incorporating Core Team edits.
ISTQB 2018	9-December-2017	Alpha review 2.5 release – Technical edit of 2.0 release, no new content added
ISTQB 2018	22-November-2017	Alpha review 2.0 release – Certified Tester Foundation Level Syllabus Major Update 2018 – see Appendix C – Release Notes for details
ISTQB 2018	12-June-2017	Alpha review release - Certified Tester Foundation Level Syllabus Major Update 2018 – see Appendix C – Release Notes
ISTQB 2011	1-Apr-2011	Certified Tester Foundation Level Syllabus Maintenance Release – see Release Notes
ISTQB 2010	30-Mar-2010	Certified Tester Foundation Level Syllabus Maintenance Release – see Release Notes
ISTQB 2007	01-May-2007	Certified Tester Foundation Level Syllabus Maintenance Release
ISTQB 2005	01-July-2005	Certified Tester Foundation Level Syllabus
ASQF V2.2	July-2003	ASQF Syllabus Foundation Level Version 2.2 “Lehrplan Grundlagen des Software-testens“
ISEB V2.0	25-Feb-1999	ISEB Software Testing Foundation Syllabus 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	25/06/2108

Ringraziamenti

Questo documento (versione 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 l’editing, i gruppi di revisione e tutte le organizzazioni nazionali per i suggerimenti e gli input forniti.

Hanno partecipato alla revisione, 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 Foundation Level International Software Testing Qualifications Board (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 revisione e tutte le organizzazioni nazionali per i suggerimenti e gli input forniti.

Gruppo di lavoro Foundation Level International Software Testing Qualifications Board (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 tutte le organizzazioni nazionali per i suggerimenti e gli input forniti.

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

Questo documento è stato formalmente rilasciato dall’Assemblea Generale dell’ISTQB® il 20 Aprile 2018.

0. Introduzione

0.1 Scopo di questo Documento

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

ISTQB fornisce questo documento come segue:

1. Ai board nazionali per tradurlo nella loro lingua locale e per accreditare i fornitori della formazione. Tali enti possono adattare il Syllabus alle loro specifiche esigenze linguistiche e aggiungere riferimenti per adattarlo alle esigenze editoriali locali.
2. Agli organismi di certificazione per ricavarne le domande d'esame nella loro lingua locale adattate agli obiettivi di apprendimento del Syllabus stesso.
3. Ai fornitori della formazione, 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

Il livello di qualificazione ‘Foundation’ è rivolto 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. Questo livello di qualifica denominato appunto ‘Foundation’ è anche appropriato per chiunque voglia costruirsi una comprensione dei fondamenti del software testing, come product owner, project manager, quality manager, software development manager, business analyst, IT director e management consultant.

I possessori del ‘Foundation Certificate’ saranno in grado di conseguire un livello di qualifica superiore nelle competenze di software testing.

L‘Overview ISTQB Livello Foundation 2018 è un documento separato che include quanto segue:

- 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 risultati aziendali e vengono utilizzati per creare gli esami di livello foundation Certified Tester.

In generale tutti i contenuti di questo Syllabus sono esaminabili a livello K1, ad eccezione dell'introduzione e delle appendici. Al candidato potrebbe essere perciò 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 menzionato esplicitamente negli obiettivi di apprendimento.

0.4 L’esame del Certificato a Livello Foundation

L'esame per il Certificato a Livello Foundation 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. Tutte 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 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 coerente 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 si intende ottenere
- Una descrizione dei concetti chiave per l’insegnamento, inclusi i riferimenti a fonti come letteratura universalmente accettata o standard

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

0.7 Come è Organizzato questo Syllabus

Esso contiene sei capitoli con contenuti esaminabili. L'intestazione di livello superiore per ogni capitolo specifica il tempo da dedicare al capitolo stesso; i tempi non sono forniti sotto il livello del capitolo. Per corsi di formazione accreditati, il Syllabus richiede un minimo di 16,75 ore di istruzione, distribuite nei sei capitoli come segue:

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

1. Fondamenti del Testing – 175 minuti

Parole Chiave

copertura, debug, difetto, errore, failure, qualità, quality assurance, root cause, analisi dei test, base di test, caso di test, completamento dei test, condizioni di test, controllo dei test, dati di test, progettazione dei test, esecuzione dei test, pianificazione dell’esecuzione dei test, implementazione dei test, monitoraggio dei test, oggetto di test, obiettivo del testing, oracolo del test, pianificazione dei test, procedura di test, test suite, test, 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 testing 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 del mantenere la tracciabilità fra base di test e prodotti di lavoro dei 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 necessaria per le attività di testing e quella per le attività di sviluppo

1.1 Cos’è il Testing

I sistemi software sono parte integrante della vita, dalle applicazioni aziendali (ad es. bancarie) ai prodotti di consumo (ad es. automobili). La maggior parte delle persone ha avuto a che fare 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 suoi malfunzionamenti.

Un'errata percezione comune è che il testing consista solo nel fare girare test, cioè nell'eseguire il software e controllare i risultati dell’esecuzione. 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 dei test, 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; questo testing è chiamato testing dinamico. Altri test non comportano l'esecuzione del componente o sistema; tale testing è chiamato testing statico. Quindi il testing include anche la revisione di prodotti di lavoro, come i requisiti, le user story e il codice sorgente.

Un'altra errata percezione comune sul testing è che esso si concentri interamente sulla verifica dei requisiti, delle user story o di altre specifiche. Poiché il testing implica la verifica della conformità del sistema ai requisiti, comprende anche la validazione, cioè il controllo sul fatto che sistema soddisfi le necessità dell'utente ed altri stakeholder negli ambienti operativi. Le attività di testing sono organizzate e svolte in modo differente nei diversi cicli di vita (si veda il paragrafo 2.1).

1.1.1 Obiettivi Tipici del Testing

Per ogni progetto gli obiettivi del testing possono includere:

- Valutare i prodotti di lavoro come requisiti, user story, progettazione e codice
- Verificare se tutti i requisiti specificati sono stati soddisfatti
- Validare se l'oggetto di test è completo e funziona come atteso da utenti e altri stakeholder
- Aumentare la fiducia nel livello di qualità dell'oggetto di test
- Prevenire i difetti
- Trovare le failure e i difetti
- Fornire informazioni sufficienti agli stakeholder, per consentire loro di prendere decisioni informate, soprattutto per quanto riguarda il livello di qualità dell'oggetto di test
- Ridurre il livello di rischio di produrre software di qualità inadeguata (ad es. failure non rilevate in precedenza che si verificano in ambiente operativo)
- Rispettare i requisiti o gli standard contrattuali, legali o regolamentari e/o verificare la conformità dell’oggetto di test a tali requisiti e 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 quante 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 diversi. L'esecuzione dei test può mostrare failure causate 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 e il testing di componente associato. Tuttavia, nello sviluppo Agile e in altri cicli di vita, 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 di componenti e sistemi, inclusa la relativa documentazione associata, può aiutare a ridurre il rischio che si verifichino failure durante il funzionamento. Quando i difetti vengono rilevati e successivamente corretti, ciò contribuisce alla qualità dei componenti e sistemi. Inoltre, il testing del software potrebbe anche essere necessario per soddisfare requisiti contrattuali o legali o standard specifici del settore.

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 delle failure o non siano soddisfatte le varie esigenze degli stakeholder. Tuttavia, utilizzando opportune tecniche di testing, si è in grado di ridurre la frequenza di tali rilasci problematici, specialmente quando queste tecniche siano applicate con il livello adeguato di skill di testing, in livelli di test opportuni e in punti adeguati nel ciclo di vita dello sviluppo software. Ad esempio:

- Avere tester coinvolti nelle revisioni dei requisiti o nel perfezionamento delle user story potrebbe consentire di rilevare difetti in questi prodotti di lavoro. L'identificazione e la rimozione dei difetti nei requisiti riduce il rischio di sviluppare funzionalità errate o non testabili.
- Avere tester che lavorano a stretto contatto con i progettisti dei sistemi durante la fase di progettazione, può aumentare la reciproca comprensione della progettazione e di come testarla. Questa migliore comprensione può ridurre il rischio di includere 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.
- Ottenere dai tester la verifica e la validazione del software prima del rilascio può consentire di rilevare failure che potrebbero altrimenti essere trascurate e può supportare il processo di rimozione dei difetti che hanno causato le failure (ad es. il debugging). Ciò aumenta la probabilità che il software soddisfi i requisiti e le esigenze degli stakeholder.

Oltre a questi esempi, il raggiungimento di obiettivi di testing 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: la gestione della qualità. La gestione della qualità comprende tutte le attività che dirigono e controllano un'organizzazione per quanto riguarda la qualità. Tra le altre attività, la gestione della qualità include sia la garanzia della qualità (la quality assurance, appunto) che il controllo della qualità. La garanzia della qualità è tipicamente focalizzata sull'aderenza ad appositi processi, al fine di fornire confidenza sul raggiungimento di livelli adeguati di qualità. Quando i processi vengono seguiti correttamente, i prodotti di lavoro creati da questi processi sono generalmente di qualità superiore e ciò contribuisce alla prevenzione dei difetti. Inoltre, sia l'uso della root cause analysis per rilevare e rimuovere le cause dei difetti che la corretta gestione dei risultati delle retrospettive per migliorare i processi, sono importanti per un efficace garanzia della qualità.

Il controllo della qualità coinvolge varie attività, comprese le attività di testing, che supportano il raggiungimento di adeguati livelli di qualità. Le attività di testing fanno parte dei processi di sviluppo o di manutenzione del software. Poiché la garanzia della qualità riguarda la corretta esecuzione dell'intero

processo, essa richiede 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 qualche altro prodotto di lavoro correlato. Un errore che porta all'introduzione di un difetto in un prodotto di lavoro può innescare un altro errore che porta all'introduzione di un difetto in un prodotto di lavoro correlato. Ad es. un errore nell’identificazione dei requisiti può portare a un difetto dei requisiti, con un conseguente errore di programmazione che porta a un difetto nel codice.

Se il codice difettoso viene eseguito, ciò potrebbe causare una failure, ma non necessariamente in tutte le circostanze. Ad es. alcuni difetti richiedono input o precondizioni molto specifici per innescare una 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, compresi 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
- Malintesi sulle interfacce interne al sistema e tra sistemi, specialmente quando tali interazioni sono in numero elevato
- Tecnologie nuove o non familiari

Oltre a quelle causate da difetti del codice, le failure possono anche essere causate 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 non previsti 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 l’introduzione di difetti simili in futuro. Concentrandosi sulle cause più significative, la root cause analysis può apportare dei miglioramenti al processo che impediscano l'introduzione di un numero significativo di difetti futuri.

Ad es. si supponga che pagamenti di interessi non corretti, a causa di una singola riga di codice errato, si traducano in reclami 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 c’è un’elevata percentuale di difetti nel calcolo degli interessi e se questi difetti sono causati da equivoci simili, i product owner potrebbero essere istruiti su come calcolare correttamente l’interesse, per ridurre tali difetti in futuro.

In questo esempio, i reclami 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-ETM Expert Level Management e ISTQB-EIT Expert Level Improving the Test Process.

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

Il testing può mostrare la presenza di difetti, ma non può provarne l’assenza. Il testing riduce la probabilità della presenza di difetti non rilevati nel software, ma anche se nessun difetto viene trovato, questa 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 possibile tranne che per casi banali. Piuttosto che tentare di testare in modo esaustivo, è necessario utilizzare l'analisi del rischio, le tecniche di testing e le priorità per concentrare gli effort di testing.

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

Per individuare tempestivamente i difetti è necessario avviare quanto prima le attività di testing statico e dinamico nel ciclo di vita dello sviluppo software. I test statici sono a volte indicati come *shift left*. Testare all'inizio del ciclo di vita dello sviluppo del 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 numero limitato di moduli solitamente contiene la maggior parte dei difetti scoperti durante i test prima del rilascio o è responsabile della maggior parte delle failure operative. I cluster dei difetti previsti e quelli effettivamente osservati in fase di test o di esercizio sono un input importante per un'analisi del rischio al fine di concentrare l’effort di testing (come menzionato nel principio 2).

Principio 5 – Attenzione al paradosso pesticida

Se gli stessi test sono ripetuti più e più volte, presumibilmente tali test non troveranno nessun nuovo difetto. Per rilevare nuovi difetti, i test esistenti e i dati di test potrebbero necessitare di modifiche e potrebbe essere necessario progettare nuovi test (i test non risultano più efficaci nel trovare difetti, proprio come i pesticidi non risultano più efficaci nell'uccidere gli insetti dopo un utilizzo protratto nel tempo). In alcuni casi, come nei test di regressione automatizzati, il paradosso pesticida ha un risultato favorevole, consistente nel 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. Per es. un software safety-critical viene testato in modo differente da una app mobile di commercio elettronico. Come altro esempio, il testing in un progetto Agile viene condotto in modo diverso rispetto al testing di un progetto con ciclo di vita 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 fallacia (cioè una falsa credenza) aspettarsi che il solo trovare e risolvere un gran numero di difetti garantisca il successo di un sistema. Ad es. testare accuratamente 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 i bisogni e le aspettative degli utenti o che risulti meno valido rispetto ad altri sistemi concorrenti.

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

1.4 Il Processo di Test

Non esiste un processo universale di testing del software, ma esistono insiemi comuni di attività di testing senza le quali il testing avrà meno probabilità di raggiungere i propri obiettivi. Questi insiemi di attività di testing costituiscono un processo di test. Un adeguato processo di test del software specifico per ogni situazione dipende da molti fattori. Ad es. quali attività di testing sono coinvolte in questo

processo, come queste attività sono implementate e quando queste attività possono essere pianificate in base alla strategia di test di un'organizzazione.

1.4.1 Processo di Test nel Contesto

I fattori di contesto 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 adottati
- 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 testing
- Prodotti di lavoro del testing
- Tracciabilità tra base di test e prodotti di lavoro del testing

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 possono agire efficacemente come key performance indicator (KPI) per guidare le attività finalizzate a dimostrare il raggiungimento degli obiettivi di testing 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 caso di test 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 state rilevate 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 Testing

Un processo di test è costituito dai seguenti principali gruppi 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 di attività è composto da attività costitutive che saranno descritte nei seguenti paragrafi. Ogni attività all'interno di ciascun gruppo può consistere a sua volta di più attività individuali, che possono variare da un progetto o da un rilascio all'altro.

Inoltre, sebbene molti di questi gruppi di attività possano apparire logicamente sequenziali, sono spesso implementati iterativamente. Ad es. lo sviluppo Agile è strutturato in piccole iterazioni di progettazione, sviluppo e test del software eseguite su base continua e supportate da una pianificazione continuativa. Perciò le attività di testing sono anch’esse svolte su base iterativa e continua all'interno di questo approccio di sviluppo. Anche nello sviluppo sequenziale, la sequenza logica delle attività comprende sovrapposizioni, combinazioni, contemporaneità od omissioni, per cui è di solito necessario personalizzare queste attività nel contesto del sistema e del progetto.

Pianificazione dei test

La pianificazione dei test comprende attività che definiscono i suoi obiettivi 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 piani di test possono essere rivisitati 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 degli avanzamenti effettivi rispetto al piano, utilizzando le metriche di monitoraggio dei test definite nel piano di test. Il controllo dei test prevede l'adozione delle azioni necessarie a soddisfare gli obiettivi del piano di test (che può essere aggiornato nel tempo). Il monitoraggio e il controllo dei test sono supportati dalla valutazione dei criteri di uscita, che sono indicati in alcuni cicli di vita come la “definition of done” (si veda il Syllabus ISTQB-AT Foundation Level Agile Tester Extension). Ad es. la valutazione dei criteri di uscita per l'esecuzione dei test nell’ambito di un determinato livello di test può includere:

- Verifica dei risultati e dei log dei test rispetto ai criteri di copertura specificati
- Valutazione del livello di qualità del componente o sistema in base a risultati e log dei test
- Determinazione della necessità di avere più test (ad es. se i test originariamente progettati per raggiungere un certo livello di copertura del rischio di prodotto non sono riusciti nell’intento, sono necessari test aggiuntivi da progettare ed eseguire)

L’avanzamento dei test rispetto al piano è comunicato agli stakeholder nei report di test, inclusi gli scostamenti dal piano e le informazioni per supportare qualsiasi decisione di interrompere i test stessi.

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 caratteristiche 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 es:
  - Specifiche dei requisiti, come requisiti di business, requisiti funzionali, requisiti di sistema, user story, epic, casi d'uso o prodotti simili che specificano il comportamento funzionale e non-funzionale desiderato di componenti o sistemi
  - Informazioni di progettazione e implementazione, come architettura di sistema o del software, diagrammi o documenti, specifiche di progettazione, flussi di chiamata, modelling diagram (ad es. 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 sistema stesso, inclusi codice, interfacce, metadati del database e query
  - Report di analisi dei rischi, che possono considerare aspetti funzionali, non-funzionali e strutturali del componente o sistema
- Valutare la base di test e gli elementi di test per identificare difetti di vario tipo, come ad es.:
  - ambiguità
  - omissioni
  - incoerenze
  - imprecisioni
  - contraddizioni
  - dichiarazioni superflue
- Identificare le funzionalità caratteristiche 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 commerciali/tecnici e i livelli di rischio
- Determinare 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 testing 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 descritte come obiettivi di test nei test charter. I test charter sono prodotti di lavoro tipici in alcune tipologie di test basate 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 un altro processo di revisione e/o il processo di test sia strettamente connesso con il processo di revisione. Tali attività di analisi dei test non solo verificano che i requisiti siano coerenti, espressi adeguatamente e completi, ma validano anche che i requisiti catturino correttamente le esigenze del cliente, dell’utente e degli altri stakeholder.

Ad es. tecniche come behavior-driven development (BDD) e acceptance test-driven development (ATDD), che richiedono la generazione di condizioni di test e casi di test dalle user story e di criteri di accettazione prima della codifica, possono verificare, validare e rilevare i difetti in user story e nei criteri di accettazione associati (si veda il Syllabus ISTQB Foundation Level Agile Tester Extension).

Progettazione dei test

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

La progettazione dei test include le seguenti attività principali:

- Progettazione e assegnazione di priorità ai casi di test e agli insiemi di casi di test
- Identificazione dei dati di test necessari per supportare le condizioni di test e i casi di test
- Progettazione dell'ambiente di test e identificazione di infrastrutture e strumenti necessari
- Determinazione della tracciabilità bidirezionale tra base di test, condizioni di test, casi di test e procedure di test (si veda il paragrafo 1.4.4)

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

Come nell’analisi dei test, anche la progettazione dei test può identificare difetti di test simili 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 casi di test in procedure di test. Quindi, cosi come la progettazione dei test risponde alla domanda "come testare?", 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:

- Sviluppo e assegnazione di priorità alle procedure di test e, potenzialmente, creazione di script di test automatizzati
- Creazione di test suite dalle procedure di test e (se presenti) da script di test automatizzati
- Organizzazione delle 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)
- Creazione dell'ambiente di test (compresi, potenzialmente, test harness, virtualizzazione dei servizi, simulatori e altri elementi dell'infrastruttura) e verifica che tutto il necessario sia stato correttamente impostato
- Preparazione dei dati di test e verifica che essi siano correttamente caricati nell'ambiente di test
- Verifica e aggiornamento della tracciabilità bidirezionale tra base di test, condizioni di test, casi di test, 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 immediatamente eseguiti nel 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. L'esecuzione dei test include le seguenti attività principali:

- Registrazione degli ID e delle versioni degli elementi di test o dell'oggetto di test, degli strumenti di test e del testware
- Esecuzione dei test manuale o tramite strumenti di esecuzione dei test
- Confronto dei risultati effettivi con i risultati attesi
- Analisi delle anomalie per stabilire le loro cause probabili (ad es. possono verificarsi failure dovute a difetti nel codice, ma possono verificarsi anche falsi positivi [si veda il paragrafo 1.2.3])
- Segnalazione di difetti in base alle failure osservate (si veda il paragrafo 5.6)
- Logging dei risultati dell'esecuzione dei test (ad es. pass, fail, bloccato)
- Ripetizione delle attività di testing a seguito di azioni intraprese per un'anomalia o come parte di testing pianificato (ad es. esecuzione di un test corretto, testing confermativo e/o testing di regressione)
- Verifica e aggiornamento della tracciabilità bidirezionale tra base di test, condizioni di test, casi di test, procedure di test e risultati dei test

Completamento dei test

Le attività di completamento dei test raccolgono dati dalle attività di testing completate per consolidare esperienza, testware e ogni altra informazione pertinente. Le attività di completamento dei test si svolgono in milestone predefinite del progetto, ad es. quando il sistema software è rilasciato, un progetto di test è completato (o cancellato), un'iterazione del progetto Agile è terminata (ad es. come parte di una retrospettiva), un livello di test è completato o una versione di manutenzione è completata.

Il completamento dei test include le seguenti attività principali:

- Verifica della chiusura di tutti i report sui difetti, inserimento delle richieste di modifica o di elementi del product backlog per eventuali difetti che rimangano irrisolti alla fine dell'esecuzione dei test
- Creazione di un test summary report da comunicare agli stakeholder
- Finalizzazione e archiviazione dell'ambiente di test, dei dati di test, dell'infrastruttura di test e di altro testware per il successivo riutilizzo
- Consegna del testware ai team di manutenzione, ad altri team di progetto e/o ad altri stakeholder che potrebbero beneficiare del loro utilizzo
- Analisi delle “lessons learned” dalle attività di testing completate, per determinare le modifiche necessarie per future iterazioni, rilasci e progetti
- Utilizzo delle informazioni raccolte per migliorare la maturità del processo di test

1.4.3 Prodotti di Lavoro del Testing

I prodotti di lavoro del testing vengono creati come parte del processo di test. Proprio come c'è una variabilità significativa nel modo con cui le organizzazioni implementano il processo di test, c'è 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 i prodotti di lavoro sono descritti in questo Syllabus e nel glossario ISTQB. Lo standard ISO (ISO/IEC/IEEE 29119-3) può anch’esso fungere da linea guida per i prodotti di lavoro del testing.

Molti dei prodotti di lavoro del testing descritti in questo paragrafo possono essere creati e gestiti utilizzando strumenti di gestione dei test e strumenti di 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 piani di test. Il piano di test include informazioni sulla base di test, a cui gli altri prodotti di lavoro saranno correlati tramite informazioni sulla tracciabilità (si veda di seguito e il paragrafo 1.4.4), nonché i criteri di uscita (o la definition of done) che verranno utilizzati durante il monitoraggio e controllo dei test. I piani di test 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 report di test, inclusi i test progress report (redatti su base continuativa e/o periodica) e i test summary report (redatti per le varie milestone). Tutti i report di test dovrebbero fornire dettagli significativi ai destinatari in termini di avanzamento dei test alla data dei report stessi, incluso il riepilogo dei risultati dell'esecuzione dei test una volta che questi siano a disposizione.

I prodotti di lavoro del monitoraggio e controllo dei test dovrebbero affrontare anche i problemi di 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 di questo Syllabus.

Prodotti di lavoro dell’analisi dei test

I prodotti di lavoro dell’analisi dei test includono le condizioni di test definite e le corrispondenti priorità, ognuna delle quali è idealmente riconducibile bidirezionalmente allo specifico elemento della base di test che copre. Per il testing esplorativo, l'analisi può comportare la creazione di 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

La progettazione dei test ha come risultato i casi di test e gli insiemi di casi di test necessari ad esercitare le condizioni di test definite nella precedente attività di analisi. Spesso è una buona pratica progettare casi di test di alto livello, senza valori specifici di dati di input e di risultati attesi. Tali casi di test di alto livello sono riutilizzabili in più cicli di test con diversi dati reali, documentando in ogni caso adeguatamente l’ambito del caso di test. Idealmente ogni caso di test è tracciabile bidirezionalmente nei confronti delle condizioni di test che copre.

La progettazione dei test comporta anche la progettazione e/o l'identificazione dei dati di test necessari, la progettazione dell’ambiente di test e l'identificazione di infrastrutture e strumenti, anche se la misura in cui questi sono documentati può variare significativamente.

Le condizioni di test definite nell'analisi dei test possono essere ulteriormente perfezionate nella progettazione dei test.

Prodotti di lavoro dell’implementazione dei test

I prodotti di lavoro dell’implementazione dei test includono:

- Procedure di test e definizione della 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 piano di test può essere dimostrato attraverso la tracciabilità bidirezionale tra le procedure di test e gli elementi specifici della base di test, attraverso i casi di test 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 virtualizzazione dei servizi e script di test 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 può variare in modo significativo.

I dati di test servono per assegnare valori concreti agli input e ai risultati attesi dei casi di test. Tali valori concreti, insieme alle indicazioni esplicite sul loro uso, trasformano il caso di test di alto livello in casi di test eseguibili di basso livello. Lo stesso caso di test di alto livello può utilizzare dati di test

diversi, se 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, anche se la misura con cui i test esplorativi (e la loro tracciabilità verso elementi specifici della base di test) sono documentati può variare in modo significativo.

Le condizioni di test definite nell'analisi dei test possono essere ulteriormente perfezionate 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 casi di test o delle procedure di test (ad es. pronto per essere eseguito, superato, fallito, bloccato, saltato deliberatamente, ecc.)
- Report dei difetti (si veda il paragrafo 5.6)
- Documentazione su quali elementi di test, oggetti di test, strumenti di test e testware siano 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 verso le procedure di test associate. Ad es. si può dire quali requisiti hanno superato tutti i rispettivi test pianificati, quali requisiti hanno dei test falliti e/o dei difetti ad essi associati e quali requisiti hanno dei test previsti ancora in attesa di essere eseguiti. Questo consente di verificare quali criteri di copertura siano stati soddisfatti e permette di riportare i risultati dei test in termini comprensibili 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 (ad es. a seguito di una retrospettiva di un progetto Agile), le richieste di modifica o gli elementi del product backlog, nonché il testware finale.

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

Come menzionato nel paragrafo 1.4.3, i prodotti di lavoro dei test 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 sopra descritto. Oltre alla valutazione della copertura dei test, una buona tracciabilità supporta:

- Analisi degli impatti delle modifiche
- Auditability del testing
- Conformità ai criteri di governance dell'IT
- Miglioramento della comprensibilità dei test progress report e dei test summary report per includere lo stato degli elementi della base di test (ad es. i requisiti che hanno superato i propri test, requisiti che hanno fallito i propri test e requisiti che hanno test in sospeso)
- Correlazione degli aspetti tecnici del testing a termini comprensibili agli stakeholder
- Disponibilità di informazioni atte a valutare la qualità del prodotto, la capacità di processo e lo stato di avanzamento del progetto rispetto agli obiettivi di business

Alcuni strumenti per la gestione dei test forniscono modelli dei prodotti di lavoro dei test che corrispondono totalmente o parzialmente ai prodotti di lavoro dei descritti in questo paragrafo. Alcune organizzazioni costruiscono 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 stesso, coinvolge esseri umani. Pertanto, la psicologia ha effetti importanti sul testing del software.

1.5.1 Psicologia Umana e Testing

L’identificazione dei difetti durante un testing statico come una revisione dei requisiti o una sessione di perfezionamento delle user story, oppure il rilevamento delle failure durante l'esecuzione di testing dinamico, possono essere percepiti come critiche al prodotto e al suo autore. Un elemento della psicologia chiamato “pregiudizio di conferma” (confirmation bias) può rendere difficile a una persona accettare informazioni che non concordino con le credenze da lui detenute. Ad es. poiché gli sviluppatori sono convinti che il proprio codice sia corretto, hanno un pregiudizio di conferma che rende loro difficile accettare l’evidenza che il codice non sia corretto. Oltre ai pregiudizi di conferma altri pregiudizi cognitivi possono rendere difficile per le persone 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 dai test contengono spesso cattive notizie.

Il risultato di questi fattori psicologici fa si che alcune persone possano 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 tester e analisti, product owner, progettisti e sviluppatori. Questo vale 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, le failure, i risultati dei test, gli avanzamenti dei test e i rischi, per costruire relazioni positive con i colleghi. Esempi di modi per instaurare una comunicazione positiva includono:

- Iniziare con la collaborazione piuttosto che con le battaglie. Va ricordato a tutti l'obiettivo comune di migliorare la qualità dei sistemi.
- Enfatizzare i benefici del testing. Ad es. 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 altri risultati in un modo che sia neutrale, incentrato sui fatti, senza criticare la persona che ha creato l'elemento difettoso. Scrivere report dei difetti che siano oggettivi e fattuali e revisionare i risultati.
- Cercare di capire come si sente l'altra persona e le ragioni per cui possa reagire negativamente all’informazione.
- Ottenere conferma che l'altra persona abbia capito ciò che è stato detto e viceversa.

Gli obiettivi tipici del testing sono stati discussi in precedenza (si veda il paragrafo 1.1). La chiara definizione della giusta serie di obiettivi del testing ha importanti implicazioni psicologiche. La maggior parte delle persone tende ad allineare i propri piani e comportamenti con gli obiettivi stabiliti dal team, dal management e da 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 di progettare e costruire un prodotto. Come discusso in precedenza, gli obiettivi del testing 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 più elevato nel prodotto.

Una mentalità riflette le ipotesi 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 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 che a contemplare ciò che potrebbe essere sbagliato in tali soluzioni. Inoltre, il pregiudizio di conferma rende difficile trovare errori nel proprio lavoro.

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 loro attività. Avere alcune delle attività di testing svolte da tester indipendenti aumenta l'efficacia del rilevamento dei difetti, il che è particolarmente importante per sistemi di grandi dimensioni, complessi o safety-critical. I tester indipendenti portano una prospettiva diversa da quella degli autori del prodotto (ad es. business analyst, product owner, progettisti e programmatori), poiché hanno pregiudizi cognitivi diversi dagli autori.

2. Il Testing nell’ambito del Ciclo di Vita dello Sviluppo Software – 100 minuti

Parole Chiave
testing di accettazione, alpha testing, beta testing, COTS (Commercial Off-The-Shelf), testing di integrazione dei componenti, testing di componente, testing confermativo, testing di accettazione contrattuale, testing funzionale, analisi degli impatti, testing di integrazione, testing di manutenzione, testing non-funzionale, testing di accettazione operativa, testing di regressione, testing di accettazione normativa, modello di sviluppo sequenziale, testing di integrazione dei sistemi, testing di sistema, base di test, caso di test, ambiente di test, livello di test, oggetto di test, obiettivo dei test, tipo di test, testing di accettazione utente, testing white-box

Obiettivi di Apprendimento per il Testing nell’ambito 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 del software e attività di testing nel ciclo di vita dello sviluppo del software

FL-2.1.2 (K1) Identificare i motivi per cui i modelli del ciclo di vita dello sviluppo del 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, nonché di approcci e responsabilità

2.3 Tipi di Test

FL-2.3.1 (K2) Confrontare test funzionali, non-funzionali e white-box

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

FL-2.3.3 (K2) Confrontare gli ambiti del testing confermativo e del testing di regressione

2.4 Testing di Manutenzione

FL-2.4.1 (K2) Riepilogare le ragioni che portano ad effettuare i test di manutenzione

FL-2.4.2 (K2) Descrivere il ruolo dell'analisi degli impatti nei test 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 svolte in ciascuna fase di un progetto di sviluppo e come le attività si relazionano tra loro in termini logici e cronologici. 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 che possano essere condotte le attività di testing adeguate.

In ogni modello del ciclo di vita dello sviluppo software, ci sono diverse caratteristiche per un buon testing:

- Per ogni attività di sviluppo, esiste una corrispondente attività di testing
- Ogni livello di test ha obiettivi del testing 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 perfezionare requisiti e progettazione e sono coinvolti nella revisione dei prodotti di lavoro (ad es. requisiti, progettazione, user story, ecc.) non appena siano disponibili delle loro bozze

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

Questo Syllabus classifica i comuni 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. Ciò significa che qualsiasi fase del processo di sviluppo dovrebbe iniziare quando la precedente fase è completata. In teoria non vi è alcuna sovrapposizione di fasi, ma nella pratica è utile avere feedback anticipati dalla fase successiva.

Nel modello Waterfall, le attività di sviluppo (ad es. analisi dei requisiti, progettazione, codifica, testing) sono completate una dopo l'altra. In questo modello le attività di testing sono svolte solo dopo che tutte le altre attività di sviluppo sono state completate.

A differenza del modello Waterfall, il modello a V integra il processo di test durante tutto il processo di sviluppo, attuando il principio del testing anticipato. Inoltre, il modello a V include livelli di test associati a ciascuna fase di sviluppo corrispondente, supportando ulteriormente il testing anticipato (si veda il paragrafo 2.2 per una disamina dei livelli di test). In questo modello l'esecuzione dei test associati a ciascun livello di test procede in sequenza, anche se in alcuni casi si verificano sovrapposizioni.

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.

Lo sviluppo incrementale comporta la definizione di requisiti, la progettazione, lo sviluppo e il testing di un sistema pezzo per pezzo, il che significa che le funzionalità del software crescono in modo incrementale. La dimensione di questi incrementi di funzionalità varia, con alcuni metodi che 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.

Lo sviluppo iterativo si adotta quando 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 consegna 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 es. da due a tre mesi) e gli incrementi di funzionalità sono corrispondentemente grandi, come ad esempio due o tre gruppi di funzionalità correlate.
- Scrum: ogni iterazione tende ad essere relativamente breve (ad es. con vari cicli di feedback di ore, giorni e poche settimane) e gli incrementi di funzionalità sono corrispondentemente piccoli, come ad esempio alcuni miglioramenti e/o due o tre nuove funzionalità.
- Kanban: implementato con o senza iterazioni di durata fissa, in grado di fornire sia una singola funzionalità sia un suo miglioramento, sino al completamento, oppure può raggruppare un insieme di funzionalità per il rilascio immediato.
- Spirale (o prototipazione): 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 metodi spesso prevedono livelli di test sovrapposti e iterati durante lo sviluppo. Idealmente, ogni funzionalità viene testata a diversi livelli di test mentre procede verso il rilascio. In alcuni casi i team utilizzano continuous delivery o continuous deployment, entrambi i quali comportano una significativa automazione di più livelli di test, come parte delle loro pipeline. Molti effort di sviluppo che utilizzano questi metodi includono anche il concetto di team auto-organizzanti (self-organizing team), i quali possono modificare le modalità con cui le attività di testing sono organizzate così come le relazioni tra tester e sviluppatori.

Questi metodi sviluppano un sistema in crescita progressiva, che può essere rilasciato agli utenti finali per singole funzionalità, su base iterativa, o nella più tradizionale modalità a major-release. Indipendentemente da come (e se) gli incrementi del software siano rilasciati agli utenti finali, il testing di regressione diventa sempre più importanti man mano che il sistema cresce.

A differenza dei modelli sequenziali, i modelli iterativi e incrementali possono fornire software utilizzabile in poche settimane o anche in pochi giorni, ma possono fornire anche il prodotto completo comprendente l’insieme 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-AT Foundation Level Agile Tester Extension, 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 adeguato di ciclo di vita dello sviluppo software dovrebbe essere selezionato e adattato in base all'obiettivo del progetto, al tipo di prodotto in corso di sviluppo, alle priorità di business (ad es. time-to-market), e ai rischi di prodotto e di progetto identificati. Ad es. lo sviluppo e il testing di un sistema amministrativo interno minore dovrebbe differire dallo sviluppo e test di un sistema safety-critical, come il sistema di controllo dei freni di un'automobile. Come altro esempio, in alcuni casi anche dei problemi organizzativi e culturali possono inibire la comunicazione tra i membri del team, il che può impedire lo sviluppo iterativo.

A seconda del contesto del progetto, potrebbe essere necessario combinare o riorganizzare i livelli e/o le attività di testing. Ad es. 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 testing di integrazione dei sistemi (ad es. integrazione con l'infrastruttura e altri sistemi) e a livello di testing di accettazione (funzionale e non-funzionale, insieme ai test di accettazione utente e ai test di accettazione operativa). Si veda il paragrafo 2.2 per una discussione sui livelli di test e il paragrafo 2.3 per una discussione sui tipi di test.

Inoltre, i modelli del ciclo di vita dello sviluppo software possono essere combinati. Ad es. un Modello a V 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 sviluppare e testare l'interfaccia utente (UI) e le funzionalità. La prototipazione può essere utilizzata all'inizio di un progetto, con un successivo modello di sviluppo incrementale adottato una volta completata la fase sperimentale.

Sistemi di Internet of Things (IoT), che consistono in molti oggetti diversi, come dispositivi, prodotti e servizi, applicano in genere modelli del ciclo di vita dello sviluppo software separati per ciascun oggetto. Questo comporta una sfida particolare nello sviluppo delle versioni del sistema IoT. Inoltre, il ciclo di vita dello sviluppo software di tali oggetti pone maggiormente l'accento sulle fasi successive del ciclo di vita, dopo che siano stati avviati all'utilizzo operativo (ad es. fasi di operate, update e decommission).

2.2 Livelli di Test

I livelli di test sono gruppi di attività di testing che sono organizzati e gestiti insieme. Ogni livello di test è un’istanza del processo di test, consistente nelle attività descritte nel paragrafo 1.4, svolte in relazione al software a un determinato livello di sviluppo (da singole unità o componenti a sistemi completi o, nel caso, 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, di riferimento per derivare i casi di test
- Oggetto di test (cioè cosa viene testato)
- Difetti e failure tipici
- Approcci e responsabilità specifici

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

2.2.1 Testing di Componente

Obiettivi del testing di componente

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

- Riduzione del rischio
- Verificare se i comportamenti funzionali e non-funzionali del componente corrispondono a quanto progettato e specificato
- Creazione della fiducia nella qualità del componente
- Individuare difetti nel componente
- Evitare che i difetti sfuggano ai 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 test di regressione di componente automatizzati svolgono un ruolo chiave nell’assicurare che le modifiche non abbiano alterato i componenti esistenti. I test di componente vengono spesso eseguiti separatamente dal resto del sistema, a seconda del sistema e del modello del ciclo di vita dello sviluppo software, il che può richiedere oggetti fittizi (mock object), virtualizzazione dei servizi, harness, stub e driver. Il testing di componente può coprire la funzionalità (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 dettagliata
- Codice

- Modello dati
- Specifiche del 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. non come descritte nelle specifiche di progettazione)
- Problemi di flusso dati
- Codice o logica non corretti
- Specifiche del componente

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

Approcci e responsabilità specifiche

Il testing di componente viene solitamente eseguito dallo sviluppatore che ha scritto il codice, il che richiede almeno l’accesso al codice in fase di test. Gli sviluppatori possono alternare lo sviluppo di componenti con la ricerca e la 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 casi di test di componente automatizzati può precedere la scrittura del codice dell’applicazione.

Ad es. lo sviluppo test-driven development (TDD) è altamente iterativo poiché si basa su cicli di: sviluppo di casi di test automatizzati, scrittura e integrazione di piccoli pezzi di codice, esecuzione dei test di componente, correzione di eventuali problemi ed infine 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.

Test-driven development è un esempio di approccio test-first. Benché test-driven development abbia avuto origine in eXtreme Programming (XP), esso si è esteso ad altre forme di Agile e anche a cicli di vita sequenziali (si veda il Syllabus ISTQB-AT Foundation Level Agile Tester Extension).

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:

- Riduzione del rischio
- Verificare se i comportamenti funzionali e non-funzionali delle interfacce corrispondono a quanto progettato e specificato
- Creazione della fiducia nella qualità delle interfacce
- Individuare difetti (che possono essere nelle interfacce stesse o all'interno dei componenti o dei sistemi)
- Evitare che i difetti sfuggano ai livelli di test più alti

Come per il testing di componente, in alcuni casi i test di regressione di integrazione automatizzati assicurano che le modifiche non abbiano alterato le interfacce, i componenti o i sistemi esistenti.

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

- Il testing di integrazione dei componenti si concentra 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 concentra su interazioni e interfacce tra sistemi, pacchetti e microservizi. Il testing di integrazione dei sistemi può anche coprire le interazioni con, utilizzando interfacce fornite da, organizzazioni esterne (ad es. servizi web). In questo caso l'organizzazione fornitrice non controlla le interfacce esterne e questo può che creare diverse sfide per il testing (ad es. garantire che i difetti bloccanti nel codice dell'organizzazione esterna siano risolti, organizzare gli ambienti di test, ecc.). I test di integrazione dei sistemi possono essere eseguiti dopo il testing di sistema o in parallelo con attività di 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:

- Software e progettazione di sistema
- Sequence diagram
- Specifiche di interfacce e di protocolli di comunicazione
- Casi d’uso
- Architettura a livello di componente o sistema
- Workflow
- Definizioni di 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 dati errata
- Sequenza o sincronizzazione errata delle chiamate di interfaccia
- Mancata corrispondenza dell'interfaccia
- Failure nella comunicazione tra componenti
- Failure di comunicazione tra i componenti non gestite o gestite in modo scorretto
- Assunzioni errate sul significato, sulle unità o sui limiti dei dati trasferiti fra componenti

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

- Strutture incoerenti dei messaggi scambiati tra sistemi
- Dati errati, dati mancanti o codifica dati errata
- Mancata corrispondenza dell'interfaccia
- Failure nella comunicazione tra sistemi
- Failure di comunicazione tra i sistemi non gestite o gestite in modo scorretto
- Assunzioni errate sul significato, sulle unità o sui limiti dei dati trasferiti fra sistemi
- Mancato rispetto delle norme di sicurezza obbligatorie

Approcci e responsabilità specifiche

I test di integrazione dei componenti e i test di integrazione dei sistemi dovrebbero concentrarsi sull'integrazione stessa. Per es. se si integra il modulo A con il modulo B, i test dovrebbero concentrarsi sulla comunicazione tra i moduli, non sulle funzionalità dei singoli moduli, che dovrebbero essere state coperte durante il testing di componente. Se si integra il sistema X con il sistema Y, i test dovrebbero concentrarsi sulla comunicazione tra i sistemi, non sulle funzionalità dei singoli sistemi, che dovrebbero

essere state coperte durante il testing di sistema. Sono utilizzabili tipi di test funzionali, non-funzionali e strutturali.

Il testing di integrazione dei componenti è spesso responsabilità degli sviluppatori. Il testing di integrazione dei sistemi è generalmente responsabilità dei tester. Idealmente, i tester che si occupano del testing di integrazione dei sistemi dovrebbero comprendere 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 necessario 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 (ovvero prevedere un numero ridotto di componenti o sistemi aggiuntivi ad ogni passo) piuttosto che "big bang" (ovvero in cui l'integrazione di tutti componenti o sistemi avviene in un unico passo). Un'analisi del rischio delle interfacce più complesse può essere d'aiuto per concentrare il testing di integrazione.

Maggiore è l’ambito dell'integrazione, più diventa difficile isolare i difetti in uno specifico componente o sistema, il che può comportare un aumento del rischio e un tempo aggiuntivo per la risoluzione dei problemi. Questo è uno dei motivi per cui la continuous integration, in cui il software è integrato componente per componente (ovvero per integrazione funzionale), è diventata una pratica comune. La continuous integration spesso include test di regressione automatizzati, idealmente a più livelli 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 esibisce durante l'esecuzione di tali attività. Gli obiettivi del testing di sistema includono:

- Riduzione del rischio
- Verificare se i comportamenti funzionali e non-funzionali del sistema corrispondono a quanto progettato e specificato
- Validare che il sistema è completo e funzionerà come previsto
- Creazione della fiducia nella qualità del sistema nel suo complesso
- Individuare difetti
- Evitare che i difetti sfuggano ai livelli di test più alti o in produzione

Per alcuni sistemi la verifica della qualità dei dati può essere un obiettivo. Come con il testing di componente e di integrazione, in alcuni casi i test di regressione automatizzati di sistema forniscono la confidenza che le modifiche non abbiano alterato le funzionalità esistenti o le attività end-to-end. Il testing di sistema spesso produce informazioni che vengono utilizzate dagli stakeholder per prendere le decisioni di 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 di sistema e software (funzionali e non-funzionali)
- Report di analisi del rischio
- Casi d’uso
- Epic e user story
- Modelli di comportamento del sistema
- State diagram
- Manuali dell’utente e del sistema

Oggetti di Test

Oggetti di test tipici del testing di sistema includono:

- Applicazioni
- Sistemi hardware/software
- Sistemi operativi
- Sistema sotto test (SUT=system under test)
- Configurazione del 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 inatteso
- Flussi di dati e/o di controllo errati all'interno del sistema
- Mancata esecuzione corretta e completa delle attività funzionali end-to-end
- Mancato funzionamento corretto del sistema negli ambienti di produzione
- Mancato funzionamento del sistema rispetto a quanto descritto nei manuali dell'utente e del sistema

Approcci e responsabilità specifiche

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) sulla base degli aspetti da testare del sistema. Ad es. è possibile creare una tabella delle decisioni per verificare se il comportamento funzionale sia quello descritto nelle regole di business.

I tester indipendenti svolgono normalmente il testing di sistema. Difetti nelle specifiche (ad es. user story mancanti, requisiti di business mal formulati, ecc.) possono causare mancanza di comprensione o disaccordo sul comportamento atteso del sistema. Tali situazioni possono causare falsi positivi e falsi negativi che, rispettivamente, fanno perdere tempo e riducono l'efficacia nel rilevamento dei difetti. Il coinvolgimento precoce dei tester nell’affinamento delle user story, le attività di testing statico, cosi come le revisioni, 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 concentra in genere sul comportamento e sulle capacità di un intero sistema o prodotto. Gli obiettivi del testing di accettazione includono:

- Creazione della fiducia nella qualità del sistema nel suo complesso
- Validare che il sistema sia completo e funzionerà 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 alla sua distribuzione e all’utilizzo da parte del cliente (utente finale). Durante il testing di accettazione possono essere trovati difetti, benché ciò non sia spesso un obiettivo, dato che trovare un numero significativo di difetti durante il testing di accettazione è considerato in alcuni casi un importante rischio di progetto. Il testing di accettazione può anche dover soddisfare requisiti legali, requisiti normativi o standard.

Forme comuni di testing di accettazione includono quanto segue:

- Testing di accettazione utente
- Testing di accettazione operativa
- Testing di accettazione contrattuale e normativa
- Alpha e beta testing

Ciascuno di essi è descritto nei seguenti quattro sottoparagrafi.

Testing di accettazione utente (UAT)

Il testing di accettazione del sistema da parte degli utenti è in genere focalizzato sulla validazione dell'idoneità all'utilizzo del sistema da parte degli utenti previsti, in un ambiente operativo reale o simulato. L'obiettivo principale è creare la fiducia che gli utenti possano utilizzare il sistema per soddisfare le loro esigenze, soddisfare i requisiti ed esercire i processi di business con difficoltà, costi e rischi minimi.

Testing di accettazione operativa (OAT)

Il testing di accettazione del sistema da parte del dipartimento di esercizio (operations) o dello staff di amministrazione dei sistemi viene svolto generalmente in un ambiente di produzione (simulato). I test si concentrano sugli aspetti operativi e possono includere:

- Backup e ripristino
- Installazione, disinstallazione e aggiornamento
- Disaster recovery
- Gestione utenti
- Attività di manutenzione
- Caricamento di dati e attività di migrazione
- Verifica di vulnerabilità alla sicurezza
- Performance testing

L'obiettivo principale del testing di accettazione operativa è quello di creare fiducia sul fatto che gli operatori o gli amministratori di sistema possano mantenere il sistema correttamente funzionante per gli utenti nell'ambiente operativo anche in condizioni eccezionali o difficili.

Testing di accettazione contrattuale e normativa

Il testing di accettazione contrattuale viene svolto in base ai criteri di accettazione del contratto per la produzione esterna di software. 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 normativa viene svolto in base alle normative che devono essere rispettate, come ad esempio norme governative, legali o di safety. Il testing di accettazione normativa è spesso svolto da utenti o tester indipendenti, a volte con l’auditing di agenzie di regolamentazione. L'obiettivo principale del testing di accettazione contrattuale e normativa è di creare fiducia sul fatto che la conformità contrattuale o normativa sia stata raggiunta.

Alpha e Beta testing

I testing alpha e beta sono in genere utilizzati dalle aziende di sviluppo di software commercial off-the-shelf (COTS), che desiderino ricevere feedback da utenti potenziali, utenti esistenti, clienti e/o operatori prima che il prodotto software venga immesso sul mercato. L’alpha testing viene svolto presso la sede dell’organizzazione di sviluppo, non dal team di sviluppo ma da clienti e/o operatori potenziali o esistenti, oppure da un team di test indipendente. Il beta testing viene eseguito, da clienti e/o operatori potenziali o esistenti, presso le proprie sedi. Il beta testing può essere condotto dopo l’alpha testing o può svolgersi senza che sia stato svolto l’alpha testing in precedenza.

Un obiettivo dei testing alpha e beta è la creazione della fiducia, tra clienti e/o operatori potenziali o esistenti, che essi possano utilizzare il sistema in condizioni normali, quotidiane e negli ambienti operativi in modo da raggiungere i loro obiettivi con difficoltà, costi e rischi minimi. 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 dell'utente o di business
- Normative, contratti legali e standard
- Casi d’uso

- Requisiti di sistema
- Documentazione del sistema o degli utenti
- Procedure di installazione
- Report di analisi del rischio

Inoltre, come base di test per derivare i casi di test relativi al testing di accettazione operativa, possono essere utilizzati uno o più dei seguenti prodotti di lavoro:

- Procedure di backup e ripristino
- Procedure di disaster recovery
- Requisiti non-funzionali
- Documentazione operativa
- Istruzioni di distribuzione e installazione
- Obiettivi di performance
- Pacchetti di database
- Standard o normative di sicurezza

Oggetti di Test

Oggetti di test tipici di ogni forma di testing di accettazione includono:

- Sistema sotto test
- Configurazione del sistema e dati di configurazione
- Processi di business per un sistema completamente integrato
- Sistemi di recovery e hot site (per continuità operativa e disaster recovery)
- Processi operativi e di manutenzione
- Moduli
- Report
- Dati di produzione esistenti e convertiti

Difetti e failure tipici

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

- I workflow del 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 alla sicurezza, inadeguata performance con carichi elevati o operazioni errate su una delle piattaforme supportate

Approcci e responsabilità specifiche

Il testing di accettazione è spesso responsabilità di clienti, utenti aziendali, product owner o operatori di 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 es:

- Il testing di accettazione di un prodotto software COTS può svolgersi quando tale prodotto è installato o integrato con altri prodotti
- Il testing di accettazione di un nuovo miglioramento funzionale può svolgersi prima del testing di sistema

Nello sviluppo iterativo, i team di progetto possono impiegare varie forme di testing di accettazione durante e alla fine di ogni iterazione, come quelli incentrati sulla verifica di una nuova funzionalità rispetto ai propri criteri di accettazione e quelli incentrati sulla validazione che una nuova funzionalità soddisfi le esigenze degli utenti. Inoltre, i test alpha e beta possono svolgersi al termine di ogni iterazione, dopo il completamento di ogni iterazione o dopo una serie di iterazioni. Anche i test di accettazione utente, di accettazione operativa, di accettazione normativa e contrattuale possono essere effettuati al termine 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 testing volte a testare caratteristiche specifiche di un sistema software o di una sua parte, sulla base di obiettivi di test specifici. Tali obiettivi possono includere:

- Valutare le caratteristiche di qualità funzionali, quali completezza, correttezza e appropriatezza
- Valutare le caratteristiche di qualità non-funzionali, quali affidabilità, performance, 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 es. confermando che i difetti sono stati corretti (testing confermativo) e cercando alterazioni involontarie nel funzionamento, risultanti da modifiche software o di ambiente (testing di regressione)

2.3.1 Testing Funzionale

Il testing funzionale di un sistema include test che valutano le funzioni che il sistema deve eseguire. I requisiti funzionali possono essere descritti in prodotti di lavoro come specifiche dei requisiti di business, epic, user story, casi d'uso 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 di componente possono essere basati su una specifica dei componenti), sebbene l'ambito sia diverso per ogni livello (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 casi di test relativi alla funzionalità del componente o del sistema (si veda il paragrafo 4.2).

La completezza del testing funzionale può essere misurata attraverso la copertura funzionale. La copertura funzionale è la misura con cui un certo tipo di elemento funzionale è stato esercitato dai test ed è espressa come percentuale del tipo (o dei tipi) di elemento coperto. Per es. 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 di test funzionali possono richiedere 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) o lo specifico ruolo a cui il software si indirizza (ad es. software di gioco per computer che fornisce intrattenimento interattivo).

2.3.2 Testing Non-Funzionale

Il testing non-funzionale di un sistema valuta le caratteristiche di software e sistemi, come usabilità, performance o sicurezza. Si faccia riferimento allo standard ISO (ISO/IEC 25010) per una classificazione delle caratteristiche di qualità di prodotti 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 deve, 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 ricavare condizioni di test e casi di test per il testing non-funzionale. Ad es. l'analisi ai valore limite può essere utilizzata per definire le condizioni di stress per 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 esercitati dai test ed è espressa come percentuale del tipo (o dei tipi) di elemento coperto. Ad es. utilizzando la tracciabilità tra test e dispositivi supportati per un'applicazione mobile, può essere calcolata la percentuale di dispositivi che sono stati esercitati dai test di compatibilità, identificando potenzialmente lacune nella copertura. La progettazione e l'esecuzione di test non-funzionali possono

richiedere 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 base di utenti (ad es. gli utenti dei sistemi di gestione delle strutture sanitarie).

Si faccia riferimento ai Syllabi ISTQB-ALTA Advanced Level Test Analyst, ISTQB-ATTA Advanced Level Technical Test Analyst, ISTQB-SEC Advanced Level Security Tester 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 casi di test in base alla struttura interna o all'implementazione del sistema. La struttura interna può includere codice, architettura, workflow e/o flussi di dati 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 con cui un certo tipo di elemento strutturale è stato esercitato dai test ed è espressa come percentuale del tipo di elemento coperto.

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

Progettazione ed esecuzione del testing white-box possono richiedere skill o conoscenze speciali, come il modo con cui il codice è sviluppato (ad es. per utilizzare gli strumenti di copertura del codice), come vengono memorizzati i dati (ad es. per valutare possibili query di database), come utilizzare gli strumenti di copertura, nonché 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 atto a confermare che le modifiche abbiano corretto il difetto o implementato correttamente la funzionalità e non abbiano causato conseguenze impreviste non desiderate.

- **Testing confermativo**: dopo aver corretto un difetto, il software può essere testato con tutti i casi di test falliti a causa del difetto, rieseguendoli sulla nuova versione del software. Il software può anche essere testato con nuovi test se, ad esempio, il difetto consistesse in mancanza di funzionalità. Come minimo i passi per riprodurre le failure causate dal difetto devono essere rieseguiti sulla nuova versione software. Lo scopo di un test confermativo è di assicurare che il difetto originale sia stato corretto con successo.
- **Testing di regressione**: è possibile che una modifica effettuata in una parte del codice, sia essa una correzione o un altro tipo di modifica, possa accidentalmente influenzare il comportamento di altre parti del codice, sia all'interno dello stesso componente, sia in altri componenti dello stesso sistema o anche in altri sistemi. Le modifiche possono includere modifiche all'ambiente, ad es. una nuova versione di un sistema operativo o di gestione del database. Tali effetti collaterali indesiderati sono chiamati regressioni. I test di regressione prevedono l'esecuzione di test per rilevare gli eventuali effetti collaterali indesiderati.

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

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

sostituiti. Le test suite di regressione vengono eseguite molte volte e in genere evolvono lentamente, pertanto il testing di regressione 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 ciò, sono forniti di seguito esempi di test funzionali, non-funzionali, white-box e 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 in base a come un componente deve calcolare l’interesse composto.
- Per il testing di integrazione dei componenti, i test sono progettati sulla base di come le informazioni dell’account catturate nell'interfaccia utente vengono passate alla logica di business.
- Per il testing di sistema, i test sono progettati in base al modo in cui i titolari di un account possono richiedere una linea di credito sui loro conti correnti.
- Per il testing di integrazione dei sistemi, i test sono progettati in base a come il sistema utilizza un microservizio esterno per controllare il punteggio di credito di un titolare di un conto.
- Per il testing di accettazione, i test sono progettati in base a come il bancario 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 fronte 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 layer di presentazione funziona 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, qualora il microservizio del punteggio del credito non risponda.
- Per il testing di accettazione, i test di usabilità sono progettati per valutare l'accessibilità dell’interfaccia 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 il modo 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 susseguirsi durante una richiesta di applicazione per una linea di credito.
- Per il testing di integrazione dei sistemi, i test sono progettati per esercitare tutti i possibili tipi di inquiry inviati al microservizio che restituisce il punteggio di credito.
- Per il testing di accettazione, i test sono progettati per coprire tutte le strutture di file di dati finanziari supportati e gli intervalli di valori dei trasferimenti da banca a banca.

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

- Per il testing di componente, vengono creati e inclusi nel framework di continuous integration, dei test di regressione automatizzati per ciascun componente.
- Per il testing di integrazione dei componenti, i test sono progettati per confermare la correttezza delle correzioni di difetti relativi all'interfaccia, non appena viene effettuato il check-in di tali correzioni 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'applicazione che interagisce con il microservizio che restituisce il punteggio di credito vengono rieseguiti giornalmente, come parte del continuous deployment di quel microservizio.
- 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 svolgere i tipi di test applicabili a ciascun livello, in particolare per il primo livello in cui si manifesta l’esigenza di un tipo di test.

2.4 Testing di Manutenzione

Dopo essere stati rilasciati in ambiente di produzione, software e sistemi devono essere manutenuti. Modifiche di diverse tipologie sono quasi inevitabili nei software e nei sistemi rilasciati, sia per correggere difetti scoperti durante l’utilizzo operativo, sia per aggiungere nuove funzionalità, sia per eliminare o modificare le funzionalità già fornite. La manutenzione è necessaria anche per preservare o migliorare le caratteristiche di qualità non-funzionali del componente o sistema nel corso della sua vita, in particolare le performance, la compatibilità, l'affidabilità, la sicurezza e la portabilità.

Quando durante la manutenzione vengono apportate modifiche, occorre eseguire i test di manutenzione, sia per valutare se le modifiche sono state apportate correttamente, che per verificare eventuali effetti collaterali (ad es. regressioni) in parti del sistema rimaste invariate (di solito la maggior parte del sistema). I test di manutenzione si concentrano sul testing delle modifiche al sistema e sul testing delle parti non modificate che potrebbero essere state alterate dalle modifiche. La manutenzione può comportare rilasci pianificati e rilasci non pianificati (hot fix).

Un rilascio di manutenzione può richiedere test di manutenzione a più livelli, utilizzando vari 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 col quale l'area del software modificata comunica con altri componenti o sistemi
- La dimensione del sistema esistente
- La dimensione della modifica

2.4.1 Motivazioni per la Manutenzione

Ci sono 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 motivi per cui è necessario effettuare tale manutenzione come segue:

- **Modifica**, come ad esempio miglioramenti pianificati (ad es. rilasci periodici), 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 di dati quando dei dati da un'altra applicazione sono migrati nel sistema che viene manutenuto
- **Ritiro**, ad es. 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 lunghi periodi di conservazione di tali dati sono obbligatori. Potrebbe anche essere necessario testare delle procedure di ripristino/recupero dei dati dopo la loro archiviazione per lunghi periodi di conservazione, nonché effettuare del testing di regressione per garantire il corretto funzionamento di funzionalità ancora in servizio.

Per i sistemi di Internet of Things (IoT), il testing di manutenzione può essere innescato dall'introduzione nel sistema globale di “cose” completamente nuove o modificate, come dispositivi hardware e servizi software. Il testing di manutenzione per tali sistemi pone particolare enfasi sul testing di integrazione a diversi livelli (ad es. livello di rete, livello di applicazione) 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 valuta le modifiche apportate in un rilascio di manutenzione per individuare le conseguenze attese, nonché i previsti e possibili effetti collaterali e per identificare le aree nel sistema 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 gli effetti collaterali che le aree del sistema impattate, eventualmente dopo aver aggiornato gli eventuali test esistenti influenzati dalle modifiche.

L'analisi degli impatti può essere effettuata prima che venga apportata una modifica, per aiutare a decidere se la modifica debba essere implementata, in base alle potenziali conseguenze su 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 casi di test 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 di dominio e/o di sistema
- È stata dedicata scarsa attenzione alla manutenibilità del software durante lo sviluppo

3. Testing Statico – 135 minuti

Parole Chiave

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

FL-3.2.1 (K2) Riepilogare le attività del processo di revisione del prodotto di lavoro

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

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

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

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

3.1 Fondamenti del Testing Statico

Contrariamente al testing dinamico, che richiede l'esecuzione del software sotto test, il testing statico fa affidamento sull'esame manuale dei prodotti di lavoro (cioè le revisioni) o sulla valutazione del codice da parte di strumenti (cioè l’analisi statica). Entrambi i tipi di testing statico valutano il codice o un altro prodotto di lavoro sotto test senza eseguire effettivamente il codice o il prodotto di lavoro stesso.

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

3.1.1 Prodotti di Lavoro che possono essere Revisionati dal Testing Statico

Quasi tutti i prodotti di lavoro possono essere esaminati utilizzando testing statico (revisioni e/o analisi statica), per es.:

- Specifiche, inclusi requisiti di business, requisiti funzionali e requisiti di sicurezza
- Epic, user story e criteri di accettazione
- Specifiche di architettura e progettazione
- Codice
- Testware, inclusi piani di test, casi di test, procedure di test e script di test automatizzati
- Guide per l'utente
- Pagine Web
- Contratti, piani di progetto, schedulazioni e budget
- Modelli, come activity diagram, che possono essere utilizzati per i test basati su modelli (si veda il Syllabus ISTQB-MBT Foundation Level Model-Based Tester Extension e Kramer 2016)

Le revisioni possono essere applicate a qualsiasi prodotto di lavoro che i partecipanti siano in grado di leggere e capire. 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 applicata anche con strumenti che valutino i prodotti di lavoro scritti in linguaggio naturale come i requisiti (ad esempio, controllando ortografia, grammatica e leggibilità).

3.1.2 Benefici del Testing Statico

Le tecniche di testing statico offrono una serie di vantaggi. Il testing statico, se applicato all'inizio del ciclo di vita dello sviluppo software, consente il rilevamento precoce dei difetti prima che venga eseguito il testing dinamico (ad es. nelle revisioni delle specifiche dei requisiti e della progettazione, nell’affinamento del product backlog, ecc.).

I difetti trovati in anticipo sono spesso molto meno costosi da rimuovere, rispetto ai difetti rilevati più avanti nel ciclo di vita, soprattutto rispetto ai difetti trovati dopo che il software è stato rilasciato e sia attivamente in uso. L’utilizzo di tecniche di testing statico per trovare difetti, e quindi correggerli tempestivamente, è quasi sempre molto più economico per l’organizzazione dell'utilizzo di testing dinamico, specialmente considerando i costi aggiuntivi associati all'aggiornamento di altri prodotti di lavoro e all'esecuzione di testing confermativo e di regressione.

Ulteriori vantaggi del testing statico possono includere:

- Rilevare e correggere difetti in modo più efficiente prima dell'esecuzione dinamica dei test
- Identificare i difetti che non sono facilmente individuabili con testing dinamico
- Prevenire difetti nella progettazione o nella codifica, scoprendo incongruenze, ambiguità, contraddizioni, omissioni, inesattezze e ridondanze nei requisiti
- Incrementare la produttività dello sviluppo (ad es. grazie a una migliorata progettazione, a codice più manutenibile)
- Ridurre costi e tempi dello sviluppo
- Ridurre costi e tempi del testing

- Ridurre il costo totale della qualità nel corso della vita del software, a causa di un minor numero di failure nel proseguo del ciclo di vita o dopo il rilascio operativo
- Migliorare la comunicazione tra i membri del team durante la partecipazione a revisioni

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 si completano a vicenda trovando diversi tipi di difetti.

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

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

Rispetto al testing dinamico, i difetti tipici che sono più facili e meno costosi da trovare e risolvere tramite testing statico includono:

- Difetti nei requisiti (ad es. incongruenze, ambiguità, contraddizioni, omissioni, inesattezze, 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 non definiti, 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. diversità delle unità di misura utilizzate dal sistema chiamante rispetto al sistema chiamato)
- Vulnerabilità della sicurezza (ad es. esposizione ai buffer overflow)
- Lacune o imprecisioni nella tracciabilità o nella copertura della base di test (ad es. test mancanti per un criterio di accettazione)

Inoltre, la maggior parte dei tipi di difetti di manutenibilità può essere rilevata solo mediante testing statico (ad es. impropria modularizzazione, scarsa riusabilità dei componenti, codice difficile da analizzare e modificare senza introdurre nuovi difetti).

3.2 Processo di Revisione

Le revisioni variano da informali a formali. Le revisioni informali sono caratterizzate dal non seguire un processo definito e non avere un output formale documentato. Le revisioni formali sono caratterizzate dalla partecipazione di un team, da risultati documentati e da procedure documentate per condurre la revisione. La formalità di un processo di revisione 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 revisionare, requisiti normativi e legali e/o la necessità di un audit trail.

Il focus di una revisione dipende dagli obiettivi concordati della revisione (ad es. trovare difetti, acquisire conoscenza, educare i partecipanti siano essi tester o nuovi membri del team, o discutere e decidere consensualmente).

3.2.1 Processo di Revisione dei Prodotti di Lavoro

Il processo di revisione comprende le seguenti attività principali:

Pianificazione

- Definire l'ambito, che include lo scopo della revisione, i documenti o le parti di documenti da esaminare e le caratteristiche di qualità da valutare
- Stimare effort e tempistiche

- Identificare le caratteristiche della revisione come il tipo di revisione, con ruoli, attività e checklist
- Scegliere le persone che partecipano alla revisione e assegnare i rispettivi ruoli
- Definire i criteri di ingresso e uscita per i tipi di revisione più formali (ad es., ispezioni)
- Verificare che i criteri di ingresso siano soddisfatti (per i tipi di revisione più formali)

Inizio revisione

- Distribuire (fisicamente o per via elettronica) il prodotto di lavoro e altro materiale, come ad es. moduli di registrazione dei problemi, 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 revisione

Revisione individuale (o preparazione individuale)

- Revisionare completamente o parzialmente il prodotto di lavoro
- Rilevare potenziali difetti, fare raccomandazioni e sollevare domande

Comunicazione e analisi dei problemi

- Comunicare i potenziali difetti identificati (ad es. in un incontro di revisione)
- Analizzare i potenziali difetti, assegnando loro proprietà e stato
- Valutare e documentare le caratteristiche di qualità
- Valutare i risultati della revisione rispetto ai criteri di uscita, per prendere una decisione (respingere, importanti cambiamenti necessari; accettare, accettare con modifiche minori)

Correzione e reporting

- Creare un report dei difetti per tutte le modifiche richieste
- Correggere i difetti trovati (normalmente fatto dall'autore) nel prodotto di lavoro revisionato
- Comunicare i difetti alla persona o al team appropriati (se trovati in un prodotto di lavoro correlato al prodotto di lavoro revisionato)
- Registrare lo stato aggiornato dei difetti (in revisioni formali), incluso il consenso dell'autore della segnalazione
- Raccogliere metriche (per tipi di revisione più formali)
- Verificare che i criteri di uscita siano soddisfatti (per tipi di revisione più formali)
- Accettare il prodotto di lavoro quando vengono raggiunti i criteri di uscita

I risultati di una revisione del prodotto di lavoro variano a seconda del tipo e della formalità della revisione, come descritto nel paragrafo 3.2.3.

3.2.2 Ruoli e Responsabilità nella Revisione Formale

Una revisione formale prevede normalmente i seguenti ruoli:

Autore

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

Management

- Pianifica la revisione
- Decide l'esecuzione delle revisioni
- Assegna personale, budget e tempo
- Monitora l'efficacia costi/benefici
- Avvia nuove attività di controllo in caso di risultati inadeguati

Facilitatore (o moderatore)

- Garantisce l'esecuzione efficace degli incontri di revisione (se tenuti)
- Media, se necessario, tra i vari punti di vista
- È spesso la persona da cui dipende il successo della revisione

Leader della Revisione

- Assume la responsabilità generale della revisione
- Decide chi sarà coinvolto e organizza tempi e luoghi di svolgimento della revisione

Revisori

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

Scriba (o registratore)

- Raccoglie i potenziali difetti riscontrati durante l'attività di revisione individuale
- Registra nuovi potenziali difetti, punti in sospeso e decisioni emersi dall’incontro di revisione (se tenuto)

In alcuni tipi di revisione, una persona può svolgere più di un ruolo e le azioni associate a ciascun ruolo possono anche variare in base al tipo di revisione. Con l'avvento però di strumenti per supportare il processo di revisione, in particolare la registrazione dei difetti, dei punti in sospeso e delle decisioni, spesso non c'è bisogno di uno scriba.

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

3.2.3 Tipi di Revisione

Sebbene le revisioni possano essere utilizzate per vari scopi, uno degli obiettivi principali è quello di scoprire i difetti. Tutti i tipi di revisione possono aiutare nella rilevazione dei difetti e il tipo di revisione selezionato deve essere basato sulle esigenze del progetto, le risorse disponibili, il tipo di prodotto e i relativi rischi, il dominio e la cultura aziendale.

Le revisioni possono essere classificate in base a vari attributi. Di seguito sono elencate le quattro tipologie di revisioni più comuni e i loro attributi associati.

Revisioni Informali (buddy check, pairing, pair review)

- Obiettivo principale: rilevare potenziali difetti
- Possibili ulteriori obiettivi: generare nuove idee o soluzioni, risolvere rapidamente problemi minori
- Non basate su un processo formale (documentato)
- Possono non richiedere un incontro di revisione
- Possono essere eseguite da un collega dell'autore (buddy check) o da più persone
- I risultati possono essere documentati
- Utilità variabile a seconda dei revisori
- Uso facoltativo di checklist
- Molto comunemente utilizzate nello sviluppo Agile

Walkthrough

- Obiettivi principali: trovare difetti, migliorare il prodotto software, considerare implementazioni alternative, valutare conformità a standard e specifiche
- Possibili ulteriori obiettivi: scambio di idee su variazioni di tecniche o di approcci, formazione dei partecipanti, raggiungimento del consenso
- La preparazione individuale prima dell’incontro di revisione è facoltativa
- L’incontro di revisione è in genere guidato dall'autore del prodotto di lavoro
- Lo scriba è obbligatorio
- L'uso di checklist è facoltativo
- Può assumere la forma di scenari, dry run o simulazioni
- Possono essere prodotte liste di difetti potenziali e report di revisione
- Può nella pratica variare da abbastanza informale a molto formale

Revisioni Tecniche

- Obiettivi principali: ottenere consenso, individuare potenziali difetti
- Possibili ulteriori obiettivi: valutare la qualità e creare fiducia nel prodotto di lavoro, generare nuove idee, motivare e consentire agli autori di migliorare i futuri prodotti di lavoro, considerando le implementazioni alternative

- I revisori dovrebbero essere colleghi tecnici dell'autore ed esperti tecnici nella stessa o in un'altra disciplina
- Preparazione individuale prima dell’incontro di revisione
- L'incontro di revisione è facoltativo, guidato idealmente da un facilitatore esperto (tipicamente non l'autore)
- Lo scriba è obbligatorio, possibilmente non l'autore
- L'uso di checklist è facoltativo
- I log dei potenziali difetti e i report di revisione sono normalmente prodotti

Ispezioni

- Obiettivo principale: individuazione di potenziali difetti, valutazione della qualità e rafforzamento della fiducia nel prodotto di lavoro, prevenzione di futuri difetti simili attraverso l'apprendimento dell'autore e la root cause analysis
- Possibili ulteriori obiettivi: motivare e consentire agli autori di migliorare i futuri prodotti di lavoro e il processo di sviluppo del software, raggiungendo 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 legga ad alta voce il prodotto di lavoro durante l’incontro di revisione)
- Preparazione individuale prima dell’incontro di revisione
- I revisori sono colleghi dell'autore o esperti di altre discipline rilevanti per il prodotto di lavoro
- Vengono utilizzati criteri di ingresso e uscita predefiniti
- Lo scriba è obbligatorio
- L'incontro di revisione è guidato da un facilitatore esperto (non dall'autore)
- L'autore non può ricoprire il ruolo di leader della revisione, lettore o scriba
- Vengono prodotti i log dei potenziali difetti e il report di revisione
- Sono raccolte metriche, utilizzate per migliorare l'intero processo di sviluppo del software, incluso il processo di ispezione

Un singolo prodotto di lavoro può essere oggetto di più di un tipo di revisione, nel qual caso l'ordine di conduzione può variare. Ad es. una revisione informale può essere effettuata prima di una revisione tecnica, per garantire che il prodotto di lavoro sia pronto per una revisione tecnica.

I tipi di revisione descritti sopra possono essere svolti come peer review, svolte da colleghi aventi un inquadramento organizzativo simile.

Le tipologie di difetti riscontrati in una revisione variano, in base soprattutto al prodotto di lavoro sottoposto a revisione.

Si veda il paragrafo 3.1.3 per esempi di difetti che possono essere trovati da revisioni in diversi prodotti di lavoro e Gilb 1993 per informazioni sui controlli formali.

3.2.4 Applicare Tecniche di Revisione

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

Ad hoc

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

Checklist-based

Una revisione checklist-based è una tecnica sistematica, in cui i revisori rilevano problemi sulla base di checklist distribuite all'inizio della revisione (ad es. dal facilitatore). Una checklist di una

revisione è composta da una serie 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 revisioni. Il vantaggio principale della tecnica checklist-based è la copertura sistematica di tipologie di difetti tipici. Durante la revisione individuale bisogna fare attenzione a non seguire pedissequamente la checklist, ma cercare anche difetti al di fuori della checklist stessa.

Scenari e dry run

In una revisione basata su scenari, i revisori dispongono di linee guida strutturate su come leggere il prodotto di lavoro. Un approccio basato su scenari supporta i revisori nello svolgimento di “dry run” sul prodotto di lavoro sulla base del suo utilizzo previsto (se il prodotto di lavoro è documentato in un formato adeguato come nei casi d'uso). Questi scenari forniscono ai revisori linee guida migliori su come identificare specifici tipi di difetti, rispetto alle semplici voci di checklist. Come nelle revisioni checklist-based, per non trascurare altri tipi di difetti (ad es. funzionalità mancanti), i revisori non dovrebbero essere vincolati agli scenari documentati.

Role-based

Una revisione role-based è una tecnica in cui i revisori valutano il prodotto di lavoro dal punto di vista dei ruoli di singoli stakeholder. I ruoli più diffusi includono tipologie specifiche di utenti finali (esperti, inesperti, senior, junior, ecc.) e ruoli specifici dell'organizzazione (amministratore degli utenti, amministratore di sistema, performance tester, ecc.).

Perspective-based

Nella lettura perspective-based, analogamente a una revisione role-based, i revisori, durante le proprie revisioni individuali, assumono i punti di vista di differenti stakeholder, che includono utenti finali, marketing, progettisti, tester o esercizio. L'utilizzo di tali punti di vista differenti porta a una maggiore profondità nella revisione individuale con meno duplicazioni dei problemi rilevati dai revisori.

Inoltre, la lettura perspective-based richiede anche che i revisori cerchino di utilizzare il prodotto di lavoro sotto revisione per generare il prodotto che loro deriverebbero da esso. Ad es. un tester, nel leggere una specifica dei requisiti, dovrebbe tentare di generare una bozza dei test di accettazione per vedere se sono state incluse tutte le informazioni necessarie. Infine, nella lettura perspective-based dovrebbero essere utilizzate delle checklist.

Studi empirici hanno dimostrato che la lettura perspective-based è la tecnica generale più efficace per la revisione di 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 dei diversi tipi di revisione.

3.2.5 Fattori di Successo per le Revisioni

Per ottenere una revisione di successo, occorre considerare sia la tipologia appropriata di revisione, sia le tecniche utilizzate. Inoltre, ci sono una serie di altri fattori che influenzano il risultato della revisione.

I fattori di successo organizzativi per le revisioni includono:

- Ogni revisione deve avere obiettivi chiari, definiti durante la pianificazione della revisione e utilizzati come criteri di uscita misurabili
- Devono essere adottate le tipologie di revisione adatte per raggiungere gli obiettivi, appropriate a tipo e livello dei prodotti di lavoro e adeguate per i partecipanti
- Qualsiasi tecnica di revisione utilizzata, come la revisione checklist-based o role-based, deve essere adatta all’efficace identificazione dei difetti nel prodotto di lavoro da revisionare
- Ogni eventuale checklist utilizzata deve indirizzare i principali rischi e deve essere sempre aggiornata
- I documenti di grandi dimensioni devono essere redatti e revisionati in piccole porzioni, in modo tale che il controllo di qualità venga esercitato fornendo agli autori un feedback sui difetti anticipato e frequente
- I partecipanti devono avere il tempo sufficiente per prepararsi
- Le revisioni devono essere programmate con adeguato preavviso

- Il management deve supportare il processo di revisione (ad es. schedulando un tempo adeguato per l’attività di revisione nella pianificazione di progetto)

I fattori di successo correlati alle persone per le revisioni includono:

- Le persone coinvolte per raggiungere gli obiettivi della revisione devono essere quelle giuste, ad es. le persone con differenti competenze o punti di vista, chi può utilizzare il documento come input del proprio lavoro
- I tester devono essere considerati dei revisori validi che possono contribuire alla revisione e imparare il prodotto di lavoro, il che consente loro di preparare test più efficaci e in anticipo
- I partecipanti devono dedicare tempo e attenzione ai dettagli
- Le revisioni devono essere condotte a piccoli blocchi, in modo che i revisori non perdano concentrazione durante la revisione individuale e/o l’incontro di revisione (se effettuato)
- I difetti riscontrati devono essere riconosciuti, apprezzati e gestiti in modo obiettivo
- L'incontro deve essere ben gestito, in modo che i partecipanti lo considerino un uso prezioso del loro tempo
- La revisione deve essere condotta in un'atmosfera di fiducia; il risultato non dovrà essere utilizzato per la valutazione dei partecipanti
- I partecipanti devono evitare il linguaggio del corpo e comportamenti che potrebbero indicare noia, esasperazione o ostilità verso altri partecipanti
- Deve essere fornita una formazione adeguata, specialmente per tipi di revisione più formali come le ispezioni
- Bisogna promuovere una cultura dell'apprendimento e del miglioramento dei processi

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

4. Tecniche di Testing – 330 minuti

Parole Chiave
tecnica di testing black-box, analisi ai valori limite, testing basato su checklist, copertura, copertura delle decisioni, testing della tabella delle decisioni, error guessing, partizionamento di equivalenza, tecnica di testing basato sull'esperienza, testing esplorativo, testing delle transizioni di stato, copertura delle istruzioni, tecnica di testing, testing dei casi d'uso, tecnica di testing white-box

Obiettivi di Apprendimento per le Tecniche di Testing

4.1 Categorie di Tecniche di Testing

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

4.2 Tecniche di Testing Black-box

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

FL-4.2.2 (K3) Applicare l'analisi ai valori limite per ricavare casi di test da requisiti specificati

FL-4.2.3 (K3) Applicare il testing della tabella delle decisioni per derivare casi di test da requisiti specificati

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

FL-4.2.5 (K2) Spiegare come derivare casi di test da un caso d’uso

4.3 Tecniche di Testing 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 l’importanza della copertura delle istruzioni e delle decisioni

4.4 Tecniche di Testing Basato sull’Esperienza

FL-4.4.1 (K2) Spiegare l'error guessing

FL-4.4.2 (K2) Spiegare il testing esplorativo

FL-4.4.3 (K2) Spiegare il testing basato su checklist

4.1 Categorie di Tecniche di Testing

Lo scopo di una tecnica di testing, comprese quelle discusse in questo paragrafo, è di aiutare l'identificazione di condizioni di test, casi di test e dati di test.

4.1.1 Scegliere le Tecniche di Testing

La scelta delle tecniche di testing da utilizzare dipende da una serie di fattori, tra cui:

- Tipo di componente o sistema
- Complessità del componente o sistema
- Standard normativi
- Requisiti del cliente o contrattuali
- Livelli di rischio
- Tipi di rischio
- Obiettivi del testing
- Documentazione disponibile
- Conoscenze e competenze dei tester
- Strumenti disponibili
- Tempo e budget
- Modello del ciclo di vita dello sviluppo del software
- Utilizzo previsto del software
- Esperienza precedente di utilizzo delle tecniche di testing sul componente o sistema da testare
- Tipologie di difetti previsti nel componente o sistema

Alcune tecniche sono applicabili meglio in determinate situazioni e a certi livelli di test, altre sono applicabili a tutti i livelli di test. Durante la creazione dei casi di test, i tester usano generalmente una combinazione di tecniche di testing per ottenere i migliori risultati dall’effort profuso.

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

4.1.2 Categorie di Tecniche di Testing e loro Caratteristiche

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

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

Le tecniche di testing 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 di quelle black-box, le tecniche di testing white-box si concentrano sulla struttura e sull'elaborazione all'interno dell'oggetto di test.

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

Caratteristiche comuni alle tecniche di testing black-box includono:

- Condizioni di test, casi di test e dati di test sono derivati da una base di test, che può includere requisiti software, specifiche, casi d'uso e user story
- I casi di test possono essere utilizzati per rilevare lacune tra requisiti e 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 testing white-box includono:

- Condizioni di test, casi di test e dati di test derivano da una base di test che può includere codice, architettura software, progettazione di dettaglio o qualsiasi altra fonte di informazione riguardante la 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)
- Le specifiche sono spesso utilizzate come fonte aggiuntiva di informazioni per determinare il risultato atteso dei casi di test

Caratteristiche comuni alle tecniche di testing basato sull'esperienza includono:

- Condizioni di test, casi di test e dati di test derivano da una base di test che può includere la conoscenza e l’esperienza di tester, sviluppatori, utenti e altri stakeholder
- Questa conoscenza ed esperienza include l'uso del software, il suo ambiente, i suoi probabili difetti e la distribuzione di questi difetti

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

4.2 Tecniche di Testing Black-box

4.2.1 Partizionamento di Equivalenza

Il partizionamento di equivalenza divide i dati in partizioni (note anche come classi di equivalenza) sulla base dell’assunzione che tutti i membri di una determinata partizione vengano elaborati allo stesso modo (si veda Kaner 2013 e Jorgensen 2014).

Esistono partizioni di equivalenza per valori validi e invalidi.

- I valori validi sono valori che devono essere accettati dal componente o sistema. Una partizione di equivalenza contenente valori validi è chiamata "partizione di equivalenza valida".
- I valori invalidi sono valori che devono essere rifiutati dal componente o sistema. Una partizione di equivalenza contenente valori invalidi è chiamata "partizione di equivalenza invalida".
- Le partizioni possono essere identificate per qualsiasi elemento dati relativo all'oggetto di test, inclusi input, output, valori interni, valori temporali (ad es. prima o dopo un evento) e parametri d’interfaccia (ad es. componenti integrati sottoposti al testing durante i test di integrazione).
- Ogni partizione può essere suddivisa, se necessario, in partizioni secondarie.
- Ogni valore deve appartenere a una sola partizione di equivalenza.
- Quando le partizioni di equivalenza invalide sono utilizzate nei casi di test, devono essere testate individualmente, cioè non combinate con altre partizioni di equivalenza invalide, per garantire che le failure non siano mascherate.

Per ottenere una copertura del 100% con questa tecnica, i casi di test devono coprire tutte le partizioni identificate (incluse le partizioni invalide) utilizzando almeno un valore di ciascuna partizione.

La copertura è 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 (Beizer 1990).

Ad es. si supponga che un campo di input accetti un singolo valore numerico e che si utilizzi una keypad per far sì che gli input non numerici siano impossibili. L'intervallo valido ha valori da 1 a 5 inclusi. Ci sono 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), i valori limite sono 6 e 9. Per la partizione invalida (valore troppo basso), c'è solo un valore limite, 0, perché la partizione ha un solo membro.

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

Il comportamento ai limiti delle partizioni di equivalenza è più probabile che sia scorretto rispetto al comportamento all'interno delle partizioni. È importante ricordare che sia i limiti specificati che quelli implementati possono essere 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 tutti questi difetti, forzando il software a mostrare comportamenti di una partizione che siano diversi da quelli 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 una serie di valori numerici (inclusi date e orari). La copertura dei valori limite per una partizione 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 tecniche di testing combinatorio sono utili per testare l'implementazione di requisiti di sistema che specificano come diverse combinazioni di condizioni si traducano in risultati diversi. Un approccio a tale metodo di testing è il 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 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, di solito con le condizioni in alto e le azioni in basso. Ogni colonna corrisponde a una regola decisionale, che definisce una combinazione unica di condizioni che determinano l'esecuzione delle azioni associate a tale regola. I valori delle condizioni e delle azioni vengono solitamente indicate con valori booleani (veri o falsi) o valori discreti (ad es. rosso, verde, blu), ma possono anche essere numeri o intervalli di numeri. Questi diversi tipi di condizioni e azioni potrebbero essere utilizzati insieme nella stessa tabella.

La notazione comune nelle tabelle decisionali è la seguente:

Per le condizioni:

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

Per le azioni:

- X indica che l'azione dovrebbe essere svolta (può anche essere indicata come Y o T o 1)
- Vuoto indica che l'azione non dovrebbe essere svolta (può anche essere indicata come - o N o F o 0)

Una tabella delle decisioni completa ha abbastanza colonne per coprire ogni combinazione di condizioni. La tabella può essere collassata eliminando le colonne contenenti combinazioni di condizioni impossibili, colonne contenenti combinazioni possibili ma non realizzabili e colonne che testano combinazioni di condizioni che non influenzano il risultato.

Per ulteriori informazioni su come collassare le tabelle delle decisioni, si veda il Syllabus ISTQB-ATA Advanced Level Test Analyst.

Lo standard minimo comune di copertura per il testing della tabella delle decisioni è avere almeno un caso di test per ogni regola decisionale nella tabella. Ciò 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 caso di test, diviso per il numero totale di regole decisionali, normalmente espressa in percentuale.

La forza del testing della tabella delle decisioni è che aiuta a identificare tutte le combinazioni di condizioni importanti, 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 a seconda delle condizioni attuali o della 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 stato. Un diagramma delle transizioni di stato mostra i possibili stati del software, nonché come il software entra, esce e passa da uno stato all'altro. Una transizione è innescata da un evento (ad es. l'input di un valore dell'utente in un campo). L'evento si traduce in una transizione. Se lo stesso evento può risultare in due o più transizioni diverse dallo stesso stato, quell'evento può essere qualificato da una condizione di guardia. Il cambiamento di stato può comportare l'azione del software (ad es. fornire in output un calcolo o un messaggio d’errore).

Una tabella delle transizioni di stato mostra tutte le transizioni valide e potenzialmente non valide tra gli stati, così come gli eventi, le condizioni di guardia e le azioni risultanti dalle transizioni valide. I diagrammi delle transizioni di stato 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 non valide.

Il testing della transizioni di stato viene utilizzato per le applicazioni basate su menu ed è ampiamente utilizzato all'interno dell’industria del software embedded. La tecnica è adatta anche per la modellazione di uno scenario di business con stati specifici o per testare la navigazione dello schermo. 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 identificati o di transizioni 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 il Syllabus ISTQB-ATA Advanced Level Test Analyst.

4.2.5 Testing dei Casi d’Uso

I test possono essere derivati da casi d'uso, che rappresentano un modo specifico di progettare le interazioni con elementi del software, incorporando i requisiti funzionali espressi dai casi d'uso stessi. I casi d'uso sono associati ad attori (utenti umani, hardware esterno, altri componenti o sistemi) e soggetti (il componente o il sistema a cui il caso d'uso è applicato).

Ogni caso d'uso specifica un comportamento che un soggetto può eseguire in collaborazione con uno o più attori (UML 2.5.1 2017). Un caso d'uso può essere descritto da interazioni e da attività, nonché da precondizioni, postcondizioni e linguaggio naturale ove appropriato. Le interazioni tra attori e soggetto possono comportare modifiche allo stato del soggetto. Le interazioni possono essere rappresentate graficamente da workflow, activity diagram o modelli di processi di business.

Un caso d'uso può includere possibili variazioni del proprio comportamento di base, inclusi comportamenti di eccezione e gestione degli errori (risposta del sistema e ripristino da errori di

programmazione, di applicazione e di comunicazione risultante ad esempio in un messaggio di errore). I test sono progettati per esercitare i comportamenti definiti nel caso d’uso (di base, di eccezione o alternativi e gestione degli errori).

La copertura può essere misurata in base al numero di comportamenti testati del caso d'uso diviso per il numero totale di comportamenti del caso d'uso, normalmente espressa in percentuale.

Per ulteriori informazioni sui criteri di copertura per il testing dei casi d'uso, si veda il Syllabus ISTQB-ATA Advanced Level Test Analyst.

4.3 Tecniche di Testing White-box

Il testing white-box si basa sulla struttura interna dell'oggetto di test. Le tecniche di testing 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 testing di componente. Esistono tecniche più avanzate che vengono utilizzate in alcuni ambienti safety-critical, mission-critical o ad alta integrità per ottenere una copertura più approfondita, ma non sono discusse qui. Per ulteriori informazioni su tali tecniche, consultare il Syllabus ISTQB Advanced Level Technical Test Analyst.

4.3.1 Testing e Copertura delle Istruzioni

Il testing delle istruzioni esercita le istruzioni 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 delle decisioni. Per fare ciò, i casi di test seguono i flussi di controllo che partono da un punto di decisione (ad es. da un'istruzione IF, un flusso per l’esito vero e uno per l’esito falso; per un'istruzione CASE sarebbero necessari casi di test per tutti gli esiti possibili, incluso quello 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 Importanza 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. Delle due tecniche white-box discusse in questo Syllabus, il testing delle istruzioni può fornire una copertura minore del testing delle decisioni.

Il raggiungimento del 100% di copertura delle decisioni, assicura l’esecuzione di tutti gli esiti delle decisioni, che includono il test dell’esito vero e anche dell’esito falso, anche quando non vi è alcuna esplicita istruzione associata all’esito falso (ad es. nel caso di un’istruzione di IF senza un ELSE nel codice). La copertura delle istruzioni aiuta a trovare difetti nel codice che non è stato esercitato da altri test. La copertura delle decisioni aiuta a trovare difetti nel codice laddove altri test non hanno seguito 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 Testing Basato sull’Esperienza

Quando si applicano tecniche di testing basato sull'esperienza, i casi di test sono derivati dagli skill e dall’intuizione del tester e dalla sua esperienza con applicazioni e tecnologie simili. Queste tecniche possono essere utili nell'individuare casi di test che non sono stati facilmente identificati da altre tecniche più sistematiche. Dipendendo dall’approccio e dall’esperienza del tester, queste tecniche possono raggiungere livelli di copertura ed efficacia molto diversi. 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

L'error guessing è una tecnica utilizzata per anticipare il verificarsi di errori, difetti e failure, basata sulla conoscenza del tester, che include:

- Come l'applicazione ha funzionato in passato
- Quali tipi di errori tendono a commettere gli sviluppatori
- Le 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 individuino tali failure e i difetti che le hanno causate. Le liste di errori, difetti e failure possono essere redatte 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, dei test informali (non predefiniti) sono progettati, eseguiti, registrati e dinamicamente valutati durante l'esecuzione dei test. I risultati dei test vengono utilizzati per acquisire ulteriori informazioni sul componente o sistema e per creare test per le aree che potrebbero richiedere maggiore testing.

Talvolta il testing esplorativo viene condotto, per meglio strutturare le attività, utilizzando il testing basato su sessioni. Nel testing basato su sessioni, il testing esplorativo viene svolto entro un intervallo di tempo predefinito e il tester usa un test charter contenente gli obiettivi di test per guidare il testing. Il tester può utilizzare degli sheet di sessione per documentare i passi eseguiti e le scoperte fatte.

Il testing esplorativo è più utile quando ci sono specifiche limitate o inadeguate o pressioni sui tempi di testing. Il testing esplorativo è utile anche per integrare altre tecniche di testing 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 Basato su Checklist

Nel testing basato su checklist, 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, ne espandono una esistente o possono utilizzare una checklist esistente senza modifiche. Tali checklist possono essere basate sull’esperienza, sulla conoscenza di ciò che è importante per l'utente o sulla comprensione del perché e del come il software può fallire.

Le checklist possono essere create per supportare vari tipi di test, inclusi i test funzionali e non-funzionali. In assenza di casi di test dettagliati, i test basati su checklist possono fornire utili linee guida e un certo grado di consistenza. Poiché si tratta normalmente di elenchi di alto livello, è probabile che si verifichi una certa variabilità nei test effettivi, col risultato di una copertura potenzialmente maggiore ma di una minore ripetibilità.

5. Gestione del Testing – 225 minuti

Parole Chiave
gestione della configurazione, gestione dei difetti, criteri di ingresso, criteri di uscita, rischio di prodotto, rischio di progetto, rischio, livello di rischio, testing basato sul rischio, approccio di test, controllo del testing, stima del testing, test manager, monitoraggio dei test, piano di test, pianificazione dei test, test progress report, strategia di test, test summary report, tester

Obiettivi di Apprendimento per la Gestione del Testing

5.1 Organizzazione del Testing

FL-5.1.1 (K2) Spiegare vantaggi e 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 del Testing

FL-5.2.1 (K2) Riepilogare l’obiettivo e il contenuto di un piano di test

FL-5.2.2 (K2) Distinguere tra varie 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 dell’assegnazione delle priorità, nonché delle dipendenze tecniche e logiche, per schedulare l’esecuzione dei test per un determinato insieme di casi di test

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 destinatari dei report dei test

5.4 Gestione della Configurazione

FL-5.4.1 (K2) Spiegare come la gestione della configurazione supporta il testing

5.5 Rischi e Testing

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

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

FL-5.5.3 (K2) Descrivere, con esempi, come l'analisi del rischio di prodotto possa influenzare l'accuratezza e la portata dei test

5.6 Gestione dei Difetti

FL-5.6.1 (K3) Redigere un report dei difetti che copra i difetti rilevati durante il testing

5.1 Organizzazione del Testing

5.1.1 Testing Indipendente

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

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

- 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
- Team di testing indipendente o comunità di utenti all'interno dell'organizzazione, che riportano gerarchicamente al project management o all’executive management
- Tester indipendenti dell'organizzazione di business o della comunità di utenti o con specializzazioni in specifiche tipologie di test, come usabilità, sicurezza, performance, conformità normativa o portabilità
- Tester indipendenti esterni all'organizzazione, che lavorano sul posto (insourcing) o fuori sede (outsourcing)

Per la maggior parte delle tipologie di progetto, di solito è meglio avere più livelli 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 controllo sulla qualità del proprio lavoro.

Il modo con cui viene implementata l'indipendenza del testing varia a seconda del modello del ciclo di vita dello sviluppo software. Ad es. nello sviluppo Agile, i tester possono far parte di un team di sviluppo. In qualche organizzazione che utilizza metodi Agile, questi tester possono essere considerati membri di un più ampio team di testing indipendente. Inoltre, in tali organizzazioni, i product owner possono eseguire test di accettazione per validare le user story alla fine di ogni iterazione.

I potenziali vantaggi dell'indipendenza del testing includono:

- I tester indipendenti sono in grado di riconoscere diversi tipi di failure rispetto agli sviluppatori, a causa dei loro diversi background, prospettive tecniche e pregiudizi
- Un tester indipendente può verificare, contestare o confutare le assunzioni formulate dagli stakeholder durante le specifiche e l'implementazione del sistema

I potenziali svantaggi dell'indipendenza del testing includono:

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

Molte organizzazioni sono in grado di ottenere con successo i vantaggi dell'indipendenza del testing evitando gli svantaggi.

5.1.2 Compiti del Test Manager e del 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 della gestione efficace delle attività di testing. Il ruolo di test manager può 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 dei quali guidato da un test leader o da un lead tester, possono riferire a un test manager, a un test coach o a un test coordinator.

Le tipiche attività del test manager possono includere:

- Sviluppare o rivedere una policy di test e una strategia di test per l'organizzazione
- Pianificare le attività di testing considerando il contesto e comprendendo obiettivi e rischi del testing. Ciò può includere la scelta degli approcci al testing, 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, nonché la pianificazione della gestione dei difetti
- Redigere e aggiornare il/i piano/i di test
- Coordinare il/i piani/o di test con project manager, product owner e altri
- Condividere le prospettive del testing con altre attività del 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 e controllare lo stato dei criteri di uscita (o “definition of done”)
- Preparare e consegnare i test progress report e i test summary report in base alle informazioni raccolte
- Adattare la pianificazione in base ai risultati e agli avanzamenti 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
- Supportare la predisposizione di un sistema di gestione dei difetti e di un'adeguata gestione della configurazione del testware
- Introdurre metriche adeguate per misurare gli avanzamenti dei test e valutare la qualità del testing e del prodotto
- Supervisionare la selezione e l'adozione di strumenti per supportare il processo di test, includendo le raccomandazioni di budget per la loro scelta (e possibilmente per l’acquisto e/o l’assistenza), allocando tempo e risorse per i progetti pilota e fornendo 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 testing all'interno dell'organizzazione
- Sviluppare gli skill e le carriere dei tester (ad es. mediante piani di formazione, valutazione delle performance, coaching, ecc.)

Il modo con cui viene svolto il ruolo di test manager varia a seconda del ciclo di vita dello sviluppo software. Ad es. nello sviluppo Agile alcuni dei compiti sopra menzionati sono gestiti dal team Agile, in particolare i compiti che riguardano il testing quotidiano, svolto spesso da un tester che lavora all'interno del team. Alcuni dei compiti che coinvolgono più team o l'intera organizzazione o 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:

- Rivedere e contribuire ai piani di test
- Analizzare, esaminare e valutare requisiti, user story, criteri di accettazione, specifiche e modelli per la testabilità (cioè la base di test)
- Identificare e documentare le condizioni di test, nonché gestire la tracciabilità tra casi di test, condizioni di test e base di test
- Progettare, configurare e verificare gli ambienti di test, spesso coordinandosi con i gruppi di amministrazione di sistema e di gestione della rete
- Progettare e implementare casi di test e procedure di test
- Preparare e acquisire dati di test
- Creare la pianificazione dettagliata dell'esecuzione dei test
- Eseguire i test, valutare i risultati e documentare le deviazioni dai risultati previsti
- 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 automazione del testing)

- Valutare le caratteristiche non-funzionali, quali performance, affidabilità, usabilità, sicurezza, compatibilità e portabilità
- Esaminare i test sviluppati da altri

Le persone che lavorano all’analisi dei test, alla progettazione dei test, alle specifiche tipologie di test o all’automazione del testing, possono essere specialisti in questi ruoli. A seconda dei rischi legati al prodotto, al progetto e a seconda del modello del ciclo di vita dello sviluppo software adottato, diverse persone possono assumere il ruolo di tester nei diversi livelli di test. Per es. a livello di testing di componente e di testing di integrazione dei componenti, il ruolo di tester è spesso svolto dagli sviluppatori. A livello di testing di accettazione, il ruolo di tester è spesso svolto da business analyst, esperti in materia e utenti. A livello di testing di sistema e di testing di integrazione dei sistemi, il ruolo di tester viene spesso svolto da un team di test indipendente. Al livello di testing di accettazione operativo, il ruolo di tester viene spesso svolto dal dipartimento di esercizio e/o dagli amministratori dei sistemi.

5.2 Pianificazione e Stima del Testing

5.2.1 Scopo e Contenuto di un Piano di Test

Un piano di test delinea le attività di testing per i progetti di sviluppo e di manutenzione. La pianificazione è influenzata dalla policy 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 testing, da obiettivi, rischi, vincoli, criticità, testabilità e disponibilità di risorse.

Man mano che i progetti e i test progrediscono, diventano disponibili più informazioni e dettagli da poter includere nel piano di test. La pianificazione dei test è un'attività continua e 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 testing dovrebbero essere utilizzati per identificare i cambiamenti dei rischi, così che la pianificazione possa essere modificata. La pianificazione può essere documentata in un piano di test master e in piani di test separati per i diversi livelli di test, come testing di sistema e testing di accettazione o per tipi di test separati, come testing di usabilità e performance testing. Le attività di pianificazione dei test possono includere i seguenti contenuti, alcuni dei quali possono essere documentati in un piano di test:

- Determinare l'ambito, gli obiettivi e i rischi del testing
- Definire l'approccio generale del testing
- Integrare e coordinare le attività di testing nelle attività del ciclo di vita del software
- Prendere decisioni su cosa testare, su persone e altre risorse necessarie per svolgere le varie attività di testing e su come tali attività verranno portate avanti
- Pianificare le attività di analisi, progettazione, implementazione, esecuzione e valutazione dei test, sia in date particolari (ad es. nello sviluppo sequenziale) o 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 testing nel budget
- Determinare livello di dettaglio e struttura della documentazione dei test (ad es. fornendo modelli o documenti di esempio)

Il contenuto dei piani di test varia e può estendersi aldilà degli argomenti sopra identificati. Esempi di piani di test possono essere trovati nello standard ISO (ISO/IEC/IEEE 29119-3).

5.2.2 Strategia e Approccio di Test

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

- **Analitica**: questo tipo di strategia di test si basa su un'analisi di qualche fattore (ad es. requisito o rischio). I test basati sul rischio sono un esempio di approccio analitico, in cui i test sono progettati e prioritizzati in base al livello di rischio.
- **Basata su modello**: in questo tipo di strategia di test, i test sono progettati sulla base di modelli di alcuni aspetti 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 di stato e modelli di crescita dell'affidabilità.
- **Metodica**: questo tipo di strategia di test si basa sull'utilizzo sistematico di alcuni insiemi predefiniti di test o di condizioni di test, come una tassonomia di tipi comuni o probabili di failure, una lista di caratteristiche importanti della qualità o standard aziendali di look-and-feel per le app mobile o le pagine web.
- **Conforme a 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 o processi e standard imposti dall'organizzazione.
- **Diretta** (o **consultiva**): questo tipo di strategia di test è guidata principalmente da consigli, suggerimenti o istruzioni di stakeholder, di esperti di dominio o esperti di tecnologia, che potrebbero essere esterni al team di test o esterni all'organizzazione stessa.
- **Avversa alla regressione**: questo tipo di strategia di test è motivata dal desiderio di evitare la regressione delle funzionalità esistenti. Essa include il riutilizzo del testware esistente (in particolare di casi di test e dati di test), un’ampia automazione dei test di regressione e l’uso di test suite standard.
- **Reattiva**: in questo tipo di strategia di test, i test sono reattivi al componente o sistema testato e agli eventi che si verificano durante l'esecuzione dei test, piuttosto che essere preprogrammati (come lo sono nelle strategie precedenti). I test sono progettati e implementati e potrebbero essere immediatamente eseguiti in risposta alle 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. Per es. il testing basato sul rischio (strategia analitica) può essere combinato con il testing esplorativo (strategia reattiva); esse si completano così a vicenda e, se usate insieme, possono ottenere test più efficaci.

Mentre la strategia di test fornisce una descrizione generale 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 scegliere tecniche, livelli di test, tipi di test e per definire criteri di ingresso e criteri di uscita (rispettivamente “definition of ready” e “definition of done”). La personalizzazione della strategia si basa su decisioni prese in relazione alla complessità e agli obiettivi del progetto, al tipo di prodotto in fase di sviluppo e all’analisi dei rischi di prodotto. L'approccio scelto dipende dal contesto e può prendere in considerazione diversi fattori, come rischi, safety, risorse e skill, tecnologia, natura del sistema (ad es. custom-built vs. COTS), obiettivi del testing e normative.

5.2.3 Criteri di Ingresso e 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 definiscano quando una data attività di testing debba iniziare e quando tale attività possa essere considerata completata. I criteri di ingresso (tipicamente la “definition of ready” nello sviluppo Agile) definiscono le precondizioni per iniziare una determinata attività di testing. 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 (tipicamente la “definition of done” nello sviluppo Agile) definiscono quali condizioni devono essere raggiunte per dichiarare completati un livello di test o un insieme di test. I criteri di ingresso e uscita dovrebbero essere definiti per ciascun livello e tipo di test e dipendono dagli obiettivi del testing.

Tipici criteri di ingresso includono:

- Disponibilità di requisiti testabili, user story e/o modelli (ad es. quando si segue una strategia di test basata su modelli)
- Disponibilità di elementi di test che hanno soddisfatto i criteri di uscita per ognuno dei livelli di test precedenti
- Disponibilità dell'ambiente di test
- Disponibilità degli strumenti di test necessari
- Disponibilità di dati di test e 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 il limite concordato
- Il numero stimato di difetti rimanenti è sufficientemente basso
- I livelli raggiunti di affidabilità, performance, usabilità, sicurezza e di altre caratteristiche di qualità rilevanti sono considerati sufficienti

Anche senza che i criteri di uscita siano soddisfatti, è probabile che le attività di testing siano ridotte a causa di indisponibilità di budget, di termine dei tempi previsti e/o di pressioni per portare il prodotto sul mercato. In tali circostanze può essere accettabile terminare comunque il testing, se gli stakeholder del progetto e i responsabili del business hanno esaminato e accettato il rischio di rilasciare il prodotto senza ulteriori test.

5.2.4 Sequenza di Esecuzione dei Test

Una volta che i casi di test e le procedure di test sono stati prodotti (con alcune procedure di test eventualmente automatizzate) e assemblati in test suite, tali suite possono essere organizzate in una sequenza di esecuzione dei test, che definisce l'ordine in cui tali test devono essere eseguiti. Tale schedulazione dovrebbe tener conto di fattori quali priorità, dipendenze, test confermativi, test di regressione ed efficienza di esecuzione dei test.

Idealmente i casi di test dovrebbero essere ordinati in base ai loro livelli di priorità, normalmente eseguendo prima i casi di test con la priorità più alta. Tuttavia questa pratica potrebbe non funzionare se i casi di test o le funzionalità testate hanno dipendenze. Se un caso di test con priorità più alta è dipendente da un caso di test con priorità più bassa, il caso di test con priorità più bassa deve essere eseguito per primo. Allo stesso modo, se ci sono dipendenze tra i casi di test, essi devono essere ordinati in modo appropriato a prescindere dalle loro priorità relative. Anche ai test confermativi e di regressione devono essere assegnate delle priorità in base all’importanza di avere un rapido feedback sulle modifiche, ma anche in questo caso possono essere presenti delle dipendenze.

In alcuni casi sono possibili diverse sequenze di test, ognuna delle quali può avere un diverso livello di efficienza. In questi casi devono essere accettati compromessi tra l'efficienza dell'esecuzione dei test e il rispetto delle priorità.

5.2.5 Fattori che Influenzano l’Effort di Testing

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

Caratteristiche del prodotto

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

- Skill 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 gravità dei difetti rilevati
- Quantità di rilavorazioni richieste

5.2.6 Tecniche di Stima del Testing

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

- La tecnica metrics-based: si stima l’effort del testing basandosi su metriche di precedenti progetti simili, o basandosi su valori tipici
- La tecnica expert-based: si stima l’effort del testing sulla base dell'esperienza dei responsabili delle attività di testing o di esperti

Ad es. nello sviluppo Agile, i grafici di burndown sono esempi dell'approccio metrics-based, poiché l’effort viene misurato e registrato e viene poi utilizzato per stimare la quantità di lavoro che il team può svolgere nella successiva iterazione; invece il “planning poker” è un esempio di approccio expert-based, in quanto i membri del team stimano l’effort necessario per fornire una certa funzionalità basandosi sulla propria esperienza (si veda il Syllabus ISTQB-AT Foundation Level Agile Tester Extension).

All'interno di progetti sequenziali, i modelli di rimozione dei difetti sono esempi dell'approccio metrics-based; in essi il numero di difetti e il tempo necessario per rimuoverli sono misurati e registrati, per poter poi fornire una base di stima per progetti futuri di natura similare; la tecnica di stima Wideband Delphi invece è un esempio di approccio expert-based, nel quale gruppi di esperti forniscono stime basate sulla propria esperienza (si veda il Syllabus ISTQB-ATM Advanced Level Test Manager).

5.3 Monitoraggio e Controllo dei Test

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

Il controllo dei test descrive le azioni migliorative o correttive adottate a seguito di informazioni e metriche raccolte e (auspicabilmente) segnalate. Le azioni possono riguardare qualsiasi attività di testing e possono influenzare qualsiasi attività del ciclo di vita del software.

Esempi di azioni di controllo dei test includono:

- Riordinare le priorità dei test quando si verifica un rischio identificato (ad es. software consegnato in ritardo)
- Modificare la schedulazione dei test, a causa della disponibilità o dell'indisponibilità di un ambiente di test o di altre risorse
- Rivalutare se un elemento di test soddisfa un criterio di entrata o di uscita, a causa di una rilavorazione

5.3.1 Metriche Usate nel Testing

Durante e alla fine delle attività di testing possono essere raccolte delle metriche 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 testing rispetto agli obiettivi

Comuni metriche di test includono:

- Percentuale del lavoro svolto nella preparazione dei casi di test rispetto a quello pianificato (o percentuale dei casi di test implementati rispetto a quelli svolti)
- Percentuale del lavoro svolto nella preparazione dell'ambiente di test rispetto a quello pianificato
- Esecuzione dei casi di test (ad es. numero di casi di test eseguiti/non eseguiti, casi di test 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 di test in termini di requisiti, user story, criteri di accettazione, rischi o codice
- Completamento delle attività, allocazione e utilizzo delle risorse e loro effort
- Costo del testing, compreso il costo confrontato con il vantaggio di trovare il prossimo difetto o il costo confrontato con il vantaggio di eseguire il prossimo test

5.3.2 Scopo, Contenuto e Destinatari dei Report di Test

Lo scopo del reporting dei test è di riassumere e comunicare le informazioni sull'attività di testing, sia durante che alla fine di un'attività di testing (ad es. un livello di test). Il report di test redatto durante un'attività di testing può essere chiamato test progress report, mentre un report redatto alla fine di un'attività di testing può essere chiamato test summary report.

Durante il monitoraggio e il controllo dei test, il test manager produce regolarmente test progress report per gli stakeholder. Oltre al contenuto comune a tutti i test progress report e test summary report, tipici test progress report possono includere anche:

- Lo stato delle attività di testing e l’avanzamento rispetto al piano
- I fattori che ostacolano 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 emette il test summary report. Questo report fornisce un riassunto dei test eseguiti, sulla base dell'ultimo test progress report e di ogni altra informazione pertinente.

I comuni test progress report e test summary report possono includere:

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

Il contenuto di un report di test varia in base a progetto, requisiti organizzativi e ciclo di vita dello sviluppo software. Ad es. un progetto complesso con molti stakeholder o un progetto regolamentato potrebbero richiedere report più dettagliati e rigorosi, rispetto a un rapido aggiornamento del software. Come altro esempio, nello sviluppo Agile i report sull’avanzamento del testing possono essere incorporati nella task board, nei summary sui difetti e nei grafici di burndown, che possono essere discussi durante l’incontro giornaliero di stand-up (si veda il Syllabus ISTQB-AT Foundation Level Agile Tester Extension).

Oltre alla personalizzazione dei report di test in base al contesto del progetto, bisogna considerare la personalizzazione di tali report in base ai loro destinatari. Il tipo e la quantità di informazioni che dovrebbero essere inclusi per destinatari di tipo tecnico potrebbero essere diversi da quelli che dovrebbero essere inclusi in un summary report per l’executive management. Nel primo caso, possono essere importanti informazioni dettagliate su tipi e trend dei difetti. Nel secondo caso, potrebbe essere più appropriato un report di alto livello (ad es. un riepilogo sullo stato dei difetti per priorità, budget, schedulazione e condizioni di test passate/fallite/non testate).

Lo standard ISO (ISO/IEC/IEEE 29119-3) si riferisce a due tipologie di report di test, i test progress report e i test completion report (detti test summary report in questo Syllabus) e contiene strutture ed esempi per ogni tipologia.

5.4 Gestione della Configurazione

Lo scopo della gestione della configurazione è di stabilire e mantenere l'integrità del componente o sistema, del testware e delle loro relazioni reciproche nell’ambito del ciclo di vita del progetto e del prodotto.

Per supportare correttamente il testing, la gestione della configurazione può dover garantire quanto segue:

- Tutti gli elementi di test sono identificati in modo univoco, sono sotto controllo di versione, ne sono tracciate le modifiche e sono correlati fra loro
- Tutti gli elementi del testware sono identificati in modo univoco, sono sotto controllo di versione, ne sono tracciate le modifiche, sono correlati fra 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 gestione della configurazione.

5.5 Rischi e Testing

5.5.1 Definizione di Rischio

Il rischio implica il possibile accadimento di un evento futuro che abbia conseguenze negative. Il livello di rischio è determinato dalla probabilità dell'evento e dal suo impatto (il danno).

5.5.2 Rischi di Prodotto e di Progetto

Il rischio di prodotto implica la possibilità che un prodotto di lavoro (ad es. una specifica, un componente, un sistema o un test) non soddisfi le esigenze dei suoi utenti e/o degli stakeholder. Quando i rischi di prodotto sono associati a specifiche caratteristiche di qualità di un prodotto (ad es. idoneità funzionale, affidabilità, performance, usabilità, sicurezza, compatibilità, manutenibilità e portabilità), sono chiamato anche rischi di qualità.

Esempi di rischi di prodotto includono:

- Il software potrebbe non svolgere le funzioni previste dalle specifiche
- Il software potrebbe non svolgere le funzioni richieste dalle esigenze dell'utente, del cliente e/o degli stakeholder
- Un'architettura di sistema potrebbe 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 potrebbero essere inadeguati per un sistema di elaborazione delle transazioni ad alte performance
- Il feedback sull'esperienza utente (UX) potrebbe non soddisfare le aspettative del prodotto

Il rischio di progetto riguarda situazioni che, qualora dovessero verificarsi, potrebbero avere un effetto negativo sul raggiungimento dei suoi obiettivi. Esempi di rischi di progetto includono:

Problemi del progetto:

- Possono verificarsi ritardi nella consegna, nel completamento delle attività o nel soddisfacimento dei criteri di uscita (o nella “definition of done”)
- Finanziamenti inadeguati a causa di stime inesatte, riallocazione di fondi a progetti a più alta priorità o taglio generale dei costi in tutta l'organizzazione
- Modifiche tardive possono comportare significative rilavorazioni

Problemi organizzativi:

- Skill, formazione e personale potrebbero non essere sufficienti
- Problemi del personale potrebbero causare conflitti e criticità
- Utenti, personale aziendale o personale esperto potrebbero non essere disponibili a causa di priorità di business in conflitto

Problemi politici:

- I tester non possono comunicare adeguatamente le loro esigenze e/o i risultati dei test
- Gli sviluppatori e/o i tester potrebbero non dar seguito alle rilevazioni identificate durante il testing e le revisioni (ad es. non migliorando le pratiche di sviluppo e test)
- Ci potrebbe essere un atteggiamento inopportuno verso il testing (ad es. non viene apprezzato il valore di trovare difetti durante il testing)

Problemi tecnici:

- I requisiti potrebbero non essere definiti sufficientemente bene
- I requisiti potrebbero non essere soddisfatti, dati i vincoli esistenti
- L'ambiente di test potrebbe non essere pronto in tempo
- La conversione dei dati, la loro migrazione e il relativo supporto di strumenti potrebbero essere in ritardo
- I punti deboli del processo di sviluppo potrebbero influire sulla consistenza o sulla qualità dei prodotti del progetto, come ad esempio progettazione, codice, configurazione, dati di test e casi di test
- Una cattiva gestione dei difetti e problemi correlati potrebbero causare un accumulo di difetti e altro technical debt

Problemi del fornitore:

- Una terza parte potrebbe non riuscire a consegnare un prodotto/servizio necessario o fallire
- Problemi contrattuali potrebbero causare problemi al progetto

I rischi di progetto possono influenzare sia le attività di sviluppo che le attività di testing. Normalmente i project manager devono gestire tutti i rischi di progetto, ma non è insolito che i test manager abbiano loro la responsabilità dei rischi di progetto correlati al testing.

5.5.3 Testing basato sul Rischio e Qualità del Prodotto

Il rischio viene utilizzato per concentrare l’effort richiesto durante il testing, decidendo dove e quando iniziare i test e identificando le aree che richiedono maggiore attenzione. Il testing viene utilizzato per ridurre la probabilità del verificarsi di un evento avverso o per ridurne l'impatto, nonché come attività di mitigazione del rischio e per fornire feedback sui rischi identificati e sui rischi residui (irrisolti).

Un approccio al testing basato sul rischio offre valide opportunità per ridurre i livelli dei rischi di prodotto. Esso comporta l'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 sul rischio di prodotto risultanti vengono utilizzate per guidare la pianificazione dei test, la specifica, la preparazione e l’esecuzione dei casi di test, nonché il monitoraggio e il controllo dei test. L’analisi preventiva dei 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 testing da utilizzare
- Determinare i livelli e i tipi di test specifici da svolgere (ad es. test di sicurezza, test di accessibilità)
- Determinare l'entità del testing da effettuare
- Dare priorità al testing nel tentativo di individuare i difetti critici il prima possibile
- Determinare se possono essere svolte attività aggiuntive rispetto al testing per ridurre il rischio (ad es. formando adeguatamente progettisti senza esperienza)

Il testing basato sul rischio si basa sulle conoscenze collettive e sulle intuizioni degli stakeholder del progetto per condurre l’analisi del rischio di prodotto. Al fine di garantire che la probabilità di una failure

di prodotto sia ridotta al minimo, le attività di gestione del rischio forniscono un approccio disciplinato per:

- Analizzare (e rivalutare su base regolare) cosa può andare storto (rischi)
- Determinare quali siano i rischi importanti da affrontare
- Implementare azioni per mitigare tali rischi
- Predisporre piani di emergenza 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 Gestione dei Difetti

Dato che uno degli obiettivi del testing è di trovare difetti, i difetti trovati durante il testing dovrebbero essere tracciati. Il modo in cui questo tracciamento viene effettuato può 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. Eventuali difetti identificati dovrebbero essere indagati e tracciati dalla loro scoperta e classificazione fino alla loro risoluzione (ad es. correzione dei difetti e testing confermativo con esito positivo, differimento a una versione successiva o accettazione come limitazione permanente del prodotto, ecc.). Al fine di gestire tutti i difetti sino alla loro risoluzione, l'organizzazione dovrebbe adottare un processo di gestione dei difetti che includa un workflow e delle regole di classificazione.

Questo processo deve essere concordato con tutti coloro che partecipano alla gestione dei difetti, inclusi progettisti, sviluppatori, tester e product owner. In alcune organizzazioni il logging e il tracciamento dei difetti possono essere molto informali.

Durante il processo di gestione dei difetti, alcuni report potrebbero rivelare dei falsi positivi e non failure reali dovute a difetti. Ad es. un test può fallire quando una connessione di rete è interrotta o va in time-out. Questo comportamento non deriva da un difetto nell'oggetto di test, ma è un'anomalia che deve essere comunque indagata. I tester dovrebbero tentare di minimizzare il numero di falsi positivi segnalati come difetti.

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

Tipici report dei difetti hanno i seguenti obiettivi:

- Fornire a sviluppatori e altri stakeholder informazioni su eventuali eventi avversi verificatisi, per permettere di identificarne gli effetti, di isolare il problema con un test minimale di riproduzione e di correggere, se necessario, il difetto o di risolvere il problema in altro modo
- 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 più testing confermativo)
- Fornire suggerimenti per il miglioramento del processo di sviluppo e di test

Un report dei difetti archiviato durante i test dinamici include in genere:

- Un identificatore
- Un titolo e un breve riassunto del difetto segnalato
- La data del report, l’organizzazione emittente e l’autore
- L’identificazione dell'elemento di test (elemento di configurazione sotto test) e l'ambiente
- La fase del ciclo di vita dello sviluppo software in cui è stato osservato il difetto
- Una descrizione del difetto per consentirne la riproduzione e la risoluzione, compresi log, dump del database, schermate e registrazioni (se effettuate durante l'esecuzione dei test)
- Risultati attesi e effettivi
- Ambito o grado di impatto (severità) del difetto nei confronti degli stakeholder
- Urgenza/priorità della correzione

- Stato del report del difetto (ad es. aperto, differito, duplicato, in attesa di risoluzione, in attesa di testing confermativo, riaperto, chiuso)
- Conclusioni, raccomandazioni e approvazioni
- Problemi globali, come ad es. altre aree che potrebbero essere interessate da una modifica derivante dal difetto
- Cronologia delle modifiche, ad es. la sequenza di azioni eseguite dai membri del team di progetto rispetto al difetto, per isolarlo, correggerlo e confermarne la risoluzione
- Riferimenti, incluso il caso di test che ha rivelato il problema.

Alcuni di questi dettagli possono essere inclusi e/o gestiti automaticamente quando si utilizzano strumenti di gestione dei difetti (ad es. l'assegnazione automatica di un identificatore, l'assegnazione e l'aggiornamento dello stato del report del difetto durante il workflow, ecc.). I difetti riscontrati durante il testing statico, in particolare nelle revisioni, sono normalmente documentati in un modo diverso, ad es. nelle note dell’incontro di revisione.

Un esempio del contenuto di un report dei difetti può essere trovato nello standard ISO (ISO/IEC/IEEE 29119-3) (che tratta i report dei difetti come incident report).

6. Strumenti a Supporto del Testing – 40 minuti

Parole Chiave
testing data-driven, testing keyword-driven, strumenti di performance testing, automazione del testing, strumenti di esecuzione dei test, strumenti di gestione dei test

Obiettivi di Apprendimento per gli Strumenti a Supporto del Testing

6.1 Considerazioni sugli Strumenti di Testing

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

FL-6.1.2 (K1) Identificare vantaggi e rischi dell'automazione del testing

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

6.2 Utilizzo Efficace degli Strumenti

FL-6.2.1 (K1) Identificare i principi più significativi per la scelta di uno strumento

FL-6.2.2 (K1) Ricordare gli obiettivi di usare 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 Testing

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

- Strumenti direttamente utilizzati nei test, come strumenti di esecuzione dei test e strumenti di preparazione dei dati di test
- Strumenti che aiutano a gestire requisiti, casi di test, procedure di test, script di test automatizzati, risultati dei test, dati di test e difetti; nonché strumenti per reporting e monitoraggio dell’esecuzione dei test
- Strumenti utilizzati per investigazione e valutazione
- Qualsiasi strumento che sia utile nel testing (in tal senso, un foglio di calcolo è anch’esso uno strumento di test)

6.1.1 Classificazione degli Strumenti di Testing

Gli strumenti di testing possono avere uno o più dei seguenti scopi a seconda del contesto:

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

Gli strumenti possono essere classificati in base a diversi criteri, come scopo, costo, modello di licenza (ad es. commerciale od open source) e tecnologia adottata. Gli strumenti sono classificati in questo syllabus secondo le attività di testing che supportano.

Alcuni strumenti supportano chiaramente solo (o principalmente) un'attività; altri possono supportare più di un’attività, ma sono classificati nell'attività con la quale sono più strettamente associati. Strumenti da un unico fornitore, specialmente quelli che sono stati progettati per funzionare insieme, possono essere forniti come suite integrata.

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

Alcuni strumenti offrono un supporto che è in genere più rivolto agli sviluppatori (ad es. strumenti utilizzati durante il testing di componente e di integrazione). Tali strumenti sono contrassegnati con "(D)" nei paragrafi seguenti.

Strumenti a supporto della gestione dei test e del testware

Gli strumenti di gestione possono essere applicati a qualsiasi attività di testing dell'intero ciclo di vita dello sviluppo software.

Esempi di strumenti che supportano la gestione dei test e del testware includono:

- Strumenti di gestione dei test e strumenti di gestione del ciclo di vita delle applicazioni (ALM=Application Lifecycle Management)
- Strumenti di gestione dei requisiti (ad es. tracciabilità degli oggetti di test)
- Strumenti di gestione dei difetti
- Strumenti di gestione della configurazione
- Strumenti di continuous integration (D)

Strumenti a supporto del testing statico

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

- Strumenti che supportano le revisioni
- 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 gestibili in fase di progettazione e implementazione dei test, compresi casi di test, procedure di test e dati di test. Esempi di tali strumenti includono:

- Strumenti di progettazione dei test
- Strumenti di model-based testing
- Strumenti di preparazione dei dati di test
- Strumenti per acceptance test-driven development (ATDD) e behavior-driven development (BDD)
- Strumenti per test-driven development (TDD) (D)

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

Strumenti a supporto del logging e dell’esecuzione dei test

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

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

Strumenti a supporto delle misure di performance e dell’analisi dinamica

Gli strumenti di misurazione delle performance e di analisi dinamica sono essenziali per supportare attività di testing di carico e performance, in quanto tali attività non possono essere svolte in modo efficace manualmente. Esempi di questi strumenti includono:

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

Strumenti a supporto di esigenze specifiche del testing

Oltre agli strumenti che supportano il processo generale di test, ci sono molti altri strumenti che supportano problemi più specifici del testing. Esempi di questi includono strumenti che si concentrano su:

- Valutazione della qualità dei dati
- Conversione e migrazione dei dati
- Testing di usabilità
- Testing di accessibilità
- Testing di localizzazione
- Testing di sicurezza
- Testing di portabilità (ad es. testing di software su più piattaforme supportate)

6.1.2 Benefici e Rischi dell’Automazione del Testing

Il semplice acquisto di uno strumento non garantisce il suo successo. Ogni nuovo strumento introdotto in un'organizzazione richiede effort per ottenere benefici reali e duraturi. L’uso di strumenti di testing porta con se potenziali benefici e opportunità, ma anche rischi. Ciò è particolarmente vero per gli strumenti di esecuzione dei test (spesso indicati come automazione del testing).

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

- Riduzione del lavoro manuale ripetitivo (ad es. esecuzione di test di regressione, setup/ teardown dell'ambiente, immissione degli stessi dati di test e verifica degli 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, 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 difetti e performance)

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

- Le aspettative sullo strumento potrebbero non essere realistiche (incluse funzionalità e facilità d'uso)
- Tempo, costi ed effort per l'introduzione iniziale di uno strumento potrebbero essere sottostimati (incluse formazione e competenze esterne)
- Tempo ed effort necessari per ottenere benefici significativi e continui dallo strumento potrebbero essere sottostimati (comprese la necessità di modifiche nel processo di test e il miglioramento continuo delle modalità di utilizzo dello strumento)
- L’effort richiesto per manutenere le risorse di test generate dallo strumento potrebbe essere sottostimato
- Essere troppo confidenti nello strumento (che viene visto in sostituzione della progettazione o dell'esecuzione dei test o che porta all’utilizzo di test automatizzati laddove il testing manuale risulterebbe migliore)
- Potrebbero essere trascurati controlli di versioning delle risorse dei test
- Potrebbero essere trascurati relazioni e problemi di interoperabilità tra strumenti critici, come ad esempio strumenti di gestione dei requisiti, di gestione della configurazione, di gestione dei difetti e strumenti provenienti da più fornitori
- Il fornitore dello strumento potrebbe cessare l'attività, ritirare lo strumento o vendere lo strumento a un altro fornitore
- Il fornitore può essere carente nel supporto, negli aggiornamenti e nella correzioni di difetti
- Un progetto open source può essere sospeso
- Una nuova piattaforma o tecnologia potrebbe non essere supportata dallo strumento
- Ci potrebbe essere una proprietà non chiara dello strumento (ad es. per il supporto, per gli aggiornamenti, ecc.)

6.1.3 Considerazioni speciali per Strumenti di Esecuzione e Gestione dei Test

Per ottenere un'implementazione regolare e di successo, ci sono un certo numero di cose che dovrebbero essere considerate quando, in un'organizzazione, si selezionano e si integrano strumenti di esecuzione e gestione dei test.

Strumenti di esecuzione dei test

Gli strumenti di esecuzione dei test eseguono gli oggetti di test utilizzando script di test automatizzati. Essi spesso richiedono un effort significativo al fine di ottenere i benefici pianificati.

Catturare i test registrando le azioni di un tester manuale sembra promettente, ma questo approccio non è applicabile a molti script di test. Uno script catturato è una rappresentazione lineare di dati di test e azioni da svolgere. Questo tipo di script potrebbe essere instabile quando si verificano eventi imprevisti. L'ultima generazione di questi strumenti, che sfrutta la tecnologia di cattura dell'immagine "smart", ha aumentato l'utilità di questa classe di strumenti, anche se gli script generati richiedono ancora manutenzione continua, in quanto l'interfaccia utente del sistema evolve nel tempo.

Un approccio di testing data-driven separa gli input dei test e i risultati attesi, solitamente in un foglio di calcolo e utilizza uno script di test più generico, in grado di leggere i dati di input ed eseguire lo stesso script di test con dati diversi. I tester che non hanno familiarità con il linguaggio di scripting, possono quindi creare nuovi dati di test per questi script predefiniti.

In un approccio di testing keyword-driven, uno script generico elabora le keyword che descrivono le azioni da intraprendere (chiamate anche “action word”), chiamando poi gli script delle keyword per processare i dati di test associati.

I tester (anche se non hanno familiarità con il linguaggio di scripting) possono perciò definire i test, usando le keyword e i dati associati, che possono così essere personalizzati per l'applicazione sotto test.

Ulteriori dettagli ed esempi sugli approcci al testing data-driven e keyword-driven sono forniti nel Syllabus ISTQB-TAE Advanced Level Test Automation Engineer, in Fewster 1999 e in Buwalda 2001.

Gli approcci di cui sopra richiedono che qualcuno abbia esperienza nel linguaggio di scripting (tester, sviluppatori o specialisti nell'automazione del testing). Indipendentemente dalla tecnica di scripting utilizzata, i risultati attesi per ogni test devono essere confrontati con i risultati effettivi, o dinamicamente (mentre il test è in esecuzione) o memorizzati per un confronto successivo (post-esecuzione).

Gli strumenti di MBT (Model-Based Testing) consentono di acquisire una specifica funzionale sotto forma di modello, come un activity diagram. Questa attività viene generalmente eseguita da un progettista di sistema. Lo strumento di MBT interpreta il modello al fine di creare specifiche di casi di test, che possono quindi essere salvate in uno di strumento di gestione dei test e/o eseguite da uno strumento di esecuzione dei test (si veda il Syllabus ISTQB-MBT Foundation Level Model-Based Testing).

Strumenti di gestione dei test

Gli strumenti di gestione dei test hanno spesso bisogno di interfacciarsi con altri strumenti o fogli di calcolo per vari motivi, compresi:

- Produrre informazioni utili in un formato che si adatti alle esigenze dell'organizzazione
- Mantenere una tracciabilità coerente verso i requisiti in uno strumento di gestione dei requisiti
- Collegarsi con le informazioni sulla versione degli oggetti di test nello strumento di gestione della configurazione

Ciò è particolarmente importante da considerare quando si utilizza uno strumento integrato (ad es. Application Lifecycle Management), che include un modulo di gestione dei test (ed a volte un sistema di gestione dei difetti), così come altri moduli (ad es. la schedulazione del progetto e le informazioni di budget) che vengono utilizzati da diversi gruppi all'interno di un'organizzazione.

6.2 Utilizzo Efficace degli Strumenti

6.2.1 Principi principali per la selezione degli strumenti

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

- Valutazione della maturità dell'organizzazione, dei suoi punti di forza e di debolezza
- Identificazione delle opportunità di miglioramento del processo di testing supportato da strumenti
- Comprensione delle tecnologie utilizzate dagli oggetti di test, al fine di selezionare uno strumento che sia compatibile con tali tecnologie
- Strumenti di build e di continuous integration già in uso all'interno dell'organizzazione, al fine di garantirne la compatibilità e l'integrazione
- Valutazione dello strumento rispetto a requisiti chiari e criteri oggettivi
- Disponibilità o meno dello strumento per un periodo di test gratuito (e per quanto tempo)
- Valutazione del fornitore (inclusi formazione, supporto e aspetti commerciali) o del supporto per gli strumenti non commerciali (ad es. open source)
- Identificazione dei requisiti interni per il supporto e l’assistenza nell'uso dello strumento
- Valutazione delle esigenze di formazione, considerando gli skill di test (e di automazione del testing) di coloro lavoreranno direttamente con lo strumento
- Considerazione dei pro e dei contro dei vari modelli di licenza (ad es. commerciale o open source)
- Stima del rapporto costi-benefici, basato su business case concreto (se necessario)

Come fase finale, dovrebbe essere eseguita una valutazione proof-of-concept per stabilire se lo strumento funziona efficacemente con il software in 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 degli strumenti e un positivo proof-of-concept, per introdurre lo strumento selezionato in un organizzazione normalmente si inizia con un progetto pilota che ha i seguenti obiettivi:

- Acquisire una conoscenza approfondita dello strumento, comprendente 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 cambiare
- Decidere le modalità standard di utilizzo, gestione, archiviazione e manutenzione dello strumento e delle risorse dei test (ad es. decidere le convenzioni di denominazione di file e test, selezionare gli standard di codifica, creare librerie e definire la modularità delle test suite)
- Valutare se i benefici saranno raggiunti a costi ragionevoli
- Scegliere le metriche che lo strumento deve raccogliere e registrare e configurare lo strumento per assicurare che tali metriche siano effettivamente raccolte e segnalate

6.2.3 Fattori di Successo per gli Strumenti

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

- Distribuire lo strumento al resto dell'organizzazione in modo incrementale
- Modificare e migliorare i processi per adattarsi all'utilizzo dello strumento
- Fornire formazione, supporto e assistenza agli utenti degli strumenti
- Definire linee guida per l'utilizzo dello strumento (ad es. standard interni per l'automazione)
- Implementare le modalità di raccolta di informazioni sull'utilizzo effettivo dello strumento
- Monitorare l’utilizzo e i vantaggi dello strumento
- Raccogliere “lesson learned” da tutti gli utenti

È inoltre importante assicurarsi che lo strumento sia integrato tecnicamente e organizzativamente nel ciclo di vita dello sviluppo software, il che può coinvolgere strutture organizzative separate, responsabili delle operazioni e/o di fornitori terzi.

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

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-MBT Foundation Level Model-Based Tester Extension Syllabus

ISTQB-AT Foundation Level Agile Tester Extension Syllabus

ISTQB-ATA Advanced Level Test Analyst Syllabus

ISTQB-ATM Advanced Level Test Manager Syllabus

ISTQB-SEC Advanced Level Security Tester Syllabus

ISTQB-TAE Advanced Level Test Automation Engineer Syllabus

ISTQB-ETM Expert Level Test Management Syllabus

ISTQB-EITP 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

Black, R., van Veenendaal, E. and Graham, D. (2012) *Foundations of Software Testing: ISTQB Certification (3e)*, Cengage Learning: London UK

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

Spillner, A., Linz, T., and Schaefer, H. (2014) *Software Testing Foundations (4e)*, Rocky Nook: San Rafael CA

8. Appendice A

8.1 Storia di questo Documento

Questo documento è il Syllabus ISTQB Certified Tester Foundation Level, il primo livello di qualificazione internazionale approvato dall'ISTQB (www.istqb.org).

Questo documento 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 revisionata dai rappresentanti di tutti i Board ISTQB, e successivamente dai professionisti provenienti dalla comunità internazionale di testing del software.

8.2 Obiettivi della Qualificazione a Livello Foundation

- Ottenere il riconoscimento del testing come essenziale e professionale specializzazione dell’ingegneria del software
- Fornire una struttura di riferimento 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 innalzare il profilo dei tester
- Promuovere pratiche di test valide e coerenti all'interno di tutte le discipline dell'ingegneria del software
- Identificare argomenti di test rilevanti e di valore per i diversi settori merceologici
- Stimolare i fornitori di software ad assumere tester certificati e ottenere così un vantaggio competitivo verso i loro concorrenti, pubblicizzando la loro politica di assunzione
- Fornire un'opportunità ai tester e a coloro che sono interessati al testing di acquisire una qualifica in materia riconosciuta a livello internazionale

8.3 Obiettivi della Qualificazione Internazionale

- Essere in grado di confrontare gli skill dei tester tra diversi paesi
- Consentire ai tester di spostarsi più facilmente attraverso i confini nazionali
- Consentire a progetti multinazionali/internazionali di avere una comprensione comune dei problemi del testing
- Aumentare il numero di tester qualificati in tutto il mondo
- Ottenere un maggiore impatto e valore con iniziative a livello internazionale piuttosto che con approcci a livello nazionale
- Sviluppare una comune struttura internazionale di conoscenze e skill di 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 paesi
- Consentire ai tester di ottenere una qualifica riconosciuta nella loro lingua madre
- Consentire la condivisione di skill e risorse tra paesi
- Fornire il riconoscimento internazionale della qualifica di tester con la partecipazione di molti paesi

8.4 Requisiti di Ingresso per la Qualificazione

L’unico criterio di ingresso per l'esame ISTQB Certified Tester Foundation Level è che i candidati abbiano un interesse per il testing del software. Tuttavia, si raccomanda vivamente ai candidati di:

- Avere almeno un background minimo nello sviluppo o nel testing del software, come ad es. sei mesi di esperienza come tester di sistema o di accettazione utente o come sviluppatore di software
- Partecipare a un corso accreditato da un board riconosciuto ISTQB

8.5 Background e Storia del Certificato Foundation 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; riorganizzati, aggiornati e estesi nel contenuto, con enfasi rivolta agli argomenti che forniscano l’aiuto più pratico ai tester.

Un certificato Foundation nel testing del software (ad es. rilasciato da ISEB, ASQF o da un board riconosciuto ISTQB) rilasciato prima della pubblicazione del presente Certificato Internazionale, sarà riconosciuto equivalente ad esso. Il certificato Foundation non scade e non ha bisogno essere rinnovato. La data di assegnazione è indicata sul certificato.

All'interno di ciascun paese partecipante, gli aspetti locali sono controllati da un board nazionale o regionale riconosciuto da ISTQB. I compiti di ogni board sono specificati da ISTQB, ma sono implementati all'interno di ciascun paese. Tali compiti comprendono l'accreditamento di fornitori della formazione e l’organizzazione degli esami.

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

I seguenti obiettivi di apprendimento sono richiesti da 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 od un concetto.

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

Esempio
Può riconoscere la definizione di “failure” come:
- “mancato delivery del servizio ad un utente finale o ad ogni altro stakeholder” oppure
- “deviazione del componente o del sistema dal suo delivery, servizio o risultato atteso”

Livello 2: Comprendere (K2)

Il candidato è in grado di fornire 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 di testing.

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

Esempio
Può spiegare la ragione del perché i test devono essere progettati il prima possibile:
- Per trovare i difetti quando essi sono meno costosi da rimuovere
- Per trovare prima i difetti più importanti/critici

Può spiegare le similitudini e le differenze tra il testing di integrazione e il testing di sistema:
- Similarità: testing di più di un componente e possono testare aspetti non-funzionali
- Differenze: il testing di integrazione si concentra su interfacce ed interazioni mentre il testing di sistema si concentra su aspetti dell’intero sistema, come l’elaborazione end-to-end

Livello 3: Applicare (K3)

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

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

Esempio
- Può identificare valori limite (boundary value) per partizioni valide ed invalide.
- Può scegliere casi di test da un diagramma delle transizioni di stato per coprire tutte le transizioni.

Riferimento (per i livelli di conoscenza degli obiettivi di apprendimento):
Anderson, L. W. and Krathwohl, D. R. (eds) (2001) *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*, Allyn & Bacon.

10. Appendice C – Note di Rilascio

Il Syllabus ISTQB Foundation 2018 è un importante aggiornamento e riscrittura della versione 2011. Per questo motivo non ci sono note di rilascio dettagliate per capitolo e paragrafo. Tuttavia è qui fornito un riassunto delle principali modifiche. Inoltre, ISTQB fornisce, in un documento separato “Note di Rilascio”, la tracciabilità tra gli obiettivi di apprendimento della versione 2011 e quelli della versione 2018 del Syllabus Foundation Level, evidenziando quali sono stati aggiunti, aggiornati o rimossi.

All'inizio del 2017 più di 550.000 persone in più di 100 paesi hanno intrapreso l’esame Foundation e oltre 500.000 erano 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, 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 e il contenuto di ogni capitolo o paragrafo (e le domande d'esame) e viceversa fra ogni capitolo o paragrafo (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 collaudate euristiche e formule provate essere consistenti in altri Syllabi ISTQB, che si basano su un'analisi degli obiettivi di apprendimento da coprire in ciascun capitolo.

Benché il Syllabus Foundation copra le migliori pratiche e tecniche che hanno dimostrato 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 anche 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 revisionato l'allineamento a tutti i Syllabi e al Glossario ISTQB. L'obiettivo è rendere questa versione più facile da leggere, capire, imparare e tradurre, per ottenere una maggiore utilità pratica e un equilibrio tra conoscenza e skill.

Per un'analisi dettagliata delle modifiche apportate in questa versione, consultare il ISTQB Certified Tester Foundation Level Overview 2018.