Certified Tester

Syllabus Foundation Level

v4.0

International Software Testing Qualifications Board

Avviso di Copyright

Storico delle review

Versione	Data	Osservazioni
CTFL v4.0	21.04.2023	CTFL v4.0 - Versione di rilascio
CTFL v3.1.1	01.07.2021	CTFL v3.1.1 - Aggiornamento del logo e del copyright
CTFL v3.1	11.11.2019	CTFL v3.1 - Versione di manutenzione con aggiornamenti minori
ISTQB 2018	27.04.2018	CTFL v3.0 – Candidata alla versione di rilascio
ISTQB 2011	1.04.2011	Release di Manutenzione del Syllabus CTFL
ISTQB 2010	30.03.2010	Release di Manutenzione del Syllabus CTFL
ISTQB 2007	01.05.2007	Release di Manutenzione del Syllabus CTFL
ISTQB 2005	01.07.2005	Syllabus Certified Tester Foundation Level v1.0
ASQF V2.2	07.2003	ASQF Syllabus Foundation Level v2.2 "Lehrplan Grundlagen des Software-testens"
ISEB V2.0	25.02.1999	Syllabus ISEB Software Testing Foundation v2.0

Traduzione italiana v4.0 a cura di ITA-STQB (www.ita-stqb.org) – info@ita-stqb.org

Revisione della traduzione italiana v4.0 a cura di: Alessandro Collino, Danilo Magli, Giorgio Pisani, Salvatore Reale, Cristina Sobrero.

Indice dei Contenuti

Ringraziamenti

Questo documento è stato formalmente rilasciato dalla General Assembly di ISTQB® del 21 aprile 2023.

È stato prodotto da un team dedicato appartenente ai Working Group (Gruppi di Lavoro) congiunti di ISTQB, Foundation Level e Agile: Laura Albert, Renzo Cerquozzi (vicepresidente), Wim Decoutere, Klaudia Dussa-Zieger, Chintaka Indikadahena, Arnika Hryszko, Martin Klonk, Kenji Onishi, Michaël Pilaeten (copresidente), Meile Posthuma, Gandhinee Rajkomar, Stuart Reid, Eric Riou du Cosquer (copresidente), Jean-François Riverin, Adam Roman, Lucjan Stapp, Stephanie Ulrich (vicepresidente), Eshraka Zakaria.

Il team ringrazia Stuart Reid, Patricia McQuaid e Leanne Howard per la loro review tecnica, il gruppo di review e i Member Board (Board Nazionali) per i loro suggerimenti e gli input forniti.

Hanno partecipato alla review, ai commenti e alla votazione di questo Syllabus: Adam Roman, Adam Scierski, Ágota Horváth, Ainsley Rood, Ale Rebon Portillo, Alessandro Collino, Alexander Alexandrov, Amanda Logue, Ana Ochoa, André Baumann, André Verschelling, Andreas Spillner, Anna Miazek, Armin Born, Arnd Pehl, Arne Becher, Attila Gyúri, Attila Kovács, Beata Karpinska, Benjamin Timmermans, Blair Mo, Carsten Weise, Chinthaka Indikadahena, Chris Van Bael, Ciaran O'Leary, Claude Zhang, Cristina Sobrero, Dandan Zheng, Dani Almog, Daniel Säther, Daniel van der Zwan, Danilo Magli, Darvay Tamás Béla, Dawn Haynes, Dena Pauletti, Dénes Medzihradszky, Doris Dötzer, Dot Graham, Edward Weller, Erhardt Wunderlich, Eric Riou Du Cosquer, Florian Fieber, Fran O'Hara, François Vaillancourt, Frans Dijkman, Gabriele Haller, Gary Mogyorodi, Georg Sehl, Géza Bujdosó, Giancarlo Tomasig, Giorgio Pisani, Gustavo Márquez Sosa, Helmut Pichler, Hongbao Zhai, Horst Pohlmann, Ignacio Trejos, Ilia Kulakov, Ine Lutterman, Ingvar Nordström, Iosif Itkin, Jamie Mitchell, Jan Giesen, Jean-Francois Riverin, Joanna Kazun, Joanne Tremblay, Joëlle Genois, Johan Klintin, John Kurowski, Jörn Münzel, Judy McKay, Jürgen Beniermann, Karol Frühauf, Katalin Balla, Kevin Kooh, Klaudia Dussa-Zieger, Klaus Erlenbach, Klaus Olsen, Krisztián Miskó, Laura Albert, Liang Ren, Lijuan Wang, Lloyd Roden, Lucjan Stapp, Mahmoud Khalaili, Marek Majernik, Maria Clara Choucair, Mark Rutz, Markus Niehammer, Martin Klonk, Márton Siska, Matthew Gregg, Matthias Hamburg, Mattijs Kemmink, Maud Schlich, May Abu-Sbeit, Meile Posthuma, Mette Bruhn-Pedersen, Michal Tal, Michel Boies, Mike Smith, Miroslav Renda, Mohsen Ekssir, Monika Stocklein Olsen, Murian Song, Nicola De Rosa, Nikita Kalyani, Nishan Portoyan, Nitzan Goldenberg, Ole Chr. Hansen, Patricia McQuaid, Patricia Osorio, Paul Weymouth, Pawel Kwasik, Peter Zimmerer, Petr Neugebauer, Piet de Roo, Radoslaw Smilgin, Ralf Bongard, Ralf Reissing, Randall Rice, Rik Marselis, Rogier Ammerlaan, Sabine Gschwandtner, Sabine Uhde, Salinda Wickramasinghe, Salvatore Reale, Sammy Kolluru, Samuel Ouko, Stephanie Ulrich, Stuart Reid, Surabhi Bellani, Szilard Szell, Tamás Gergely, Tamás Horváth, Tatiana Sergeeva, Tauhida Parveen, Thaer Mustafa, Thomas Eisbrenner, Thomas Harms, Thomas Heller, Tobias Letzkus, Tomas Rosenqvist, Werner Lieblang, Yaron Tsubery, Zhenlei Zuo e Zsolt Hargitai.

Working Group ISTQB Foundation Level (Edizione 2018): Klaus Olsen (presidente), Tauhida Parveen (vicepresidente), Rex Black (project manager), Eshraka Zakaria, Debra Friedenberg, Ebbe Munk, Hans Schaefer, Judy McKay, Marie Walsh, Meile Posthuma, Mike Smith, Radoslaw Smilgin, Stephanie Ulrich, Steve Toms, Corne Kruger, Dani Almog, Eric Riou du Cosquer, Igal Levi, Johan Klintin, Kenji Onishi, Rashed Karim, Stevan Zivanovic, Sunny Kwon, Thomas Müller, Vipul Kocher, Yaron Tsubery. Il Gruppo di lavoro ringrazia il Gruppo di review e tutti i Member Board per i loro suggerimenti.

Working Group ISTQB Foundation Level (Edizione 2011): Thomas Müller (presidente), Debra Friedenberg. Il Gruppo di lavoro ringrazia il gruppo di review (Dan Almog, Armin Beer, Rex Black, Julie Gardiner, Judy McKay, Tuula Pääkkönen, Eric Riou du Cosquier Hans Schaefer, Stephanie Ulrich, Erik van Veenendaal) e tutti i Member Board per i suggerimenti sulla versione attuale del Syllabus.

Working Group ISTQB Foundation Level (Edizione 2010): Thomas Müller (presidente), Rahul Verma, Martin Klonk e Armin Beer. Il Gruppo di lavoro ringrazia il gruppo di review (Rex Black, Mette Bruhn-Pederson, Debra Friedenberg, Klaus Olsen, Judy McKay, Tuula Pääkkönen, Meile Posthuma, Hans Schaefer, Stephanie Ulrich, Pete Williams, Erik van Veenendaal) e tutti i Member Board per i loro suggerimenti.

Working Group ISTQB Foundation Level (Edizione 2007): Thomas Müller (presidente), Dorothy Graham, Debra Friedenberg e Erik van Veenendaal. Il Gruppo di lavoro ringrazia il gruppo di review (Hans Schaefer, Stephanie Ulrich, Meile Posthuma, Anders Pettersson e Wonil Kwon) e tutti i Member Board per i loro suggerimenti.

Working Group ISTQB Foundation Level (Edizione 2005): Thomas Müller (presidente), Rex Black, Sigrid Eldh, Dorothy Graham, Klaus Olsen, Maaret Pyhäjärvi, Geoff Thompson e Erik van Veenendaal. Il Gruppo di lavoro ringrazia il gruppo di review e tutti i Member Board per i loro suggerimenti.

0. Introduzione

0.1. Scopo di questo Syllabus

Questo Syllabus costituisce la base per l'International Software Testing Qualification a livello Foundation. ISTQB® fornisce questo Syllabus come segue:

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

0.2. Il Certified Tester Foundation Level nel Software Testing

La certificazione Foundation Level è rivolta a chiunque sia coinvolto nel testing del software. Questo include persone che ricoprono ruoli come tester, test analyst, test engineer, consulenti del test, test manager, sviluppatori software e membri del team di sviluppo. Questa certificazione Foundation Level è appropriata anche a chiunque desideri una comprensione di base del testing del software, come project manager, quality manager, product owner, responsabili dello sviluppo software, business analyst, direttori IT e consulenti manageriali.

Solo i possessori della certificazione Foundation Level saranno abilitati al conseguimento delle certificazioni di livello superiore per il testing del software.

0.3. Percorso di Carriera per i Tester

Lo schema ISTQB® fornisce supporto ai professionisti del testing in tutte le fasi della loro carriera, offrendo sia ampiezza che profondità di conoscenze.

Chi ha conseguito la certificazione ISTQB® Foundation Level può essere interessato alle Certificazioni Core Advanced Level (Test Analyst, Technical Test Analyst e Test Manager) ed Expert Level (Test Management o Migliorare il Processo di Test).

Chiunque voglia sviluppare competenze nelle pratiche del testing in un ambiente Agile potrebbe prendere in considerazione le certificazioni Agile Technical Tester o Agile Test Leadership at Scale.

Il percorso di certificazione Specialist offre un approfondimento in aree che hanno specifici approcci di test e attività di test (es. test automation, testing dell'intelligenza artificiale, testing model-based, testing di applicazioni mobile), che sono correlate a specifiche aree di test (es., performance testing, testing di usabilità, testing di accettazione, testing di sicurezza), o che raggruppano il know-how del testing per determinati domini industriali (es., automotive o gaming).

È possibile visitare il sito www.istqb.org per rimanere aggiornati sulle ultime informazioni relative allo schema di certificazione ISTQB.

0.4. Business Outcome

Questo paragrafo elenca i 14 Business Outcome (Risultati di Business) previsti per una persona che ha conseguito la certificazione Foundation Level.

Un tester con la certificazione Foundation Level è in grado di ...

FL-BO1	Comprendere che cosa è il testing e perché è vantaggioso
FL-BO2	Comprendere i concetti fondamentali del testing del software
FL-BO3	Identificare l'approccio del test e le attività di test da implementare in base al contesto del testing.
FL-BO4	Valutare e migliorare la qualità della documentazione
FL-BO5	Aumentare l'efficacia e l'efficienza del testing
FL-BO6	Allineare il processo di test con il ciclo di vita dello sviluppo del software
FL-BO7	Comprendere i principi di test management
FL-BO8	Scrivere e comunicare defect report chiari e comprensibili
FL-BO9	Comprendere i fattori che influenzano le priorità e gli effort relativi al testing
FL-BO10	Lavorare come parte di un team cross-funzionale
FL-BO11	Conoscere i rischi e i benefici legati alla test automation
FL-BO12	Identificare le competenze principali richieste per il testing
FL-BO13	Comprendere l'impatto del rischio sul testing
FL-BO14	Fornire report in modo efficace sull'avanzamento e sulla qualità dei test

0.5. Obiettivi di Apprendimento Esaminabili e Livelli di Conoscenza

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

In generale, tutti i contenuti dei capitoli 1-6 di questo Syllabus sono esaminabili a livello K1. Al candidato può quindi essere chiesto di riconoscere, ricordare o richiamare una parola chiave o un concetto menzionato in uno qualsiasi dei sei capitoli.

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

- K1: Ricordare
- K2: Capire
- K3: Applicare

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

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

0.6. L'Esame per il Certificato Foundation Level

L'esame per il Certificato Foundation Level si basa su questo Syllabus. Le risposte alle domande d'esame possono richiedere l'uso di materiale basato su più di un paragrafo di questo Syllabus. Tutti i paragrafi del Syllabus sono esaminabili, ad eccezione dell'Introduzione e delle Appendici. Gli standard e i libri sono inclusi come riferimenti (Capitolo 7), ma il loro contenuto non è esaminabile, al di là di quanto sintetizzato in questo Syllabus relativamente a tali standard e libri. Fare riferimento al documento *Foundation Level Examination Structures and Rules*.

0.7. Accreditamento

Un Member Board ISTQB® può accreditare Training Provider, il cui materiale didattico segue questo Syllabus. I Training Provider dovrebbero ottenere le linee guida per l'accreditamento dal Member Board o dall'ente che effettua l'accreditamento. Un corso accreditato viene riconosciuto essere conforme a questo Syllabus ed è autorizzato a gestire un esame ISTQB® come parte del corso stesso. Le linee guida per l'accreditamento di questo Syllabus seguono le *Accreditation Guidelines* pubblicate dal Working Group *Processes Management and Compliance*.

0.8. Gestione degli Standard

Il Foundation Syllabus contiene riferimenti a standard (es. standard IEEE o ISO). Questi riferimenti forniscono un framework (come i riferimenti a ISO 25010 sulle caratteristiche di qualità) o forniscono una fonte di informazioni aggiuntive, se il lettore lo desidera. Il contenuto degli standard non è oggetto di esame. Per ulteriori informazioni sugli standard, fare riferimento al capitolo 7.

0.9. Rimanere Aggiornati

L'industria del software cambia rapidamente. Per far fronte a questi cambiamenti e per fornire agli stakeholder l'accesso a informazioni rilevanti e aggiornate, i Working Group ISTQB hanno creato dei link sul sito web https://www.istqb.org, che fanno riferimento alla documentazione di supporto e alle modifiche agli standard. Queste informazioni non sono oggetto di esame nell'ambito del Syllabus Foundation.

0.10. Livello di Dettaglio

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

- Obiettivi didattici generali che descrivono i propositi del Livello Foundation.
- Una lista di termini (parole chiave) che gli studenti devono essere in grado di ricordare.
- Obiettivi di apprendimento per ogni area di conoscenza, che descrivono i risultati cognitivi di apprendimento che devono essere raggiunti.
- Una descrizione dei concetti chiave, che include i riferimenti a fonti riconosciute.

Il contenuto del Syllabus non è una descrizione dell'intera area di conoscenza del testing del software; riflette il livello di dettaglio che deve essere coperto dai corsi di formazione Foundation Level. Si focalizza sui concetti e sulle tecniche di test che possono essere applicati a tutti i progetti software, indipendentemente dal ciclo di vita dello sviluppo software utilizzato.

0.11. Come è Organizzato questo Syllabus

Sono presenti sei capitoli con contenuti esaminabili. Nell'intestazione principale di ogni capitolo è specificato il tempo di formazione da dedicare al capitolo stesso. La tempistica non viene fornita per i sottocapitoli interni ad ogni capitolo. Per i corsi di formazione accreditati, il Syllabus richiede un minimo di 1135 minuti (18 ore e 55 minuti) di istruzione, distribuiti nei sei capitoli come segue:

- Capitolo 1: Fondamenti del Testing (180 minuti)
    - Lo studente apprende i principi di base relativi al testing, le ragioni per cui il testing è necessario e cosa sono gli obiettivi del testing.
    - Lo studente comprende il processo di test, le principali attività di test e il testware.
    - Lo studente comprende le competenze fondamentali per il testing.
- Capitolo 2: Testing all'interno del Ciclo di Vita dello Sviluppo Software (130 minuti)
    - Lo studente apprende come il testing viene incorporato nei diversi approcci di sviluppo.
    - Lo studente apprende i concetti degli approcci test-first e DevOps.
    - Lo studente apprende i differenti livelli di test, i tipi di test e il testing di manutenzione.
- Capitolo 3: Testing Statico (80 minuti)
    - Lo studente apprende le basi del testing statico, il processo di feedback e di review.
- Capitolo 4: Analisi e Progettazione dei Test (390 minuti)
    - Lo studente apprende come applicare le tecniche di test black-box, white-box e basate sull'esperienza, per derivare i test case da vari prodotti di lavoro software.
    - Lo studente apprende l'approccio del test basato sulla collaborazione.
- Capitolo 5: Gestione delle Attività di Test (335 minuti)

- Lo studente apprende come pianificare i test in generale, e come stimare l'effort dei test.
- Lo studente apprende come i rischi possono influenzare l'ambito del testing.
- Lo studente apprende come monitorare e controllare le attività di test.
- Lo studente apprende come il configuration management supporta il testing.
- Lo studente apprende come eseguire il reporting dei difetti in modo chiaro e comprensibile.

- Capitolo 6: Strumenti di Test (20 minuti)
  - Lo studente apprende come classificare gli strumenti per il testing, e quali sono i rischi e i benefici della test automation.

1. Fondamenti del Testing - 180 minuti

Parole Chiave

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

Obiettivi di Apprendimento per il Capitolo 1:

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 (K1) Richiamare la relazione tra testing e quality assurance

FL-1.2.3 (K2) Distinguere tra root cause, errore, difetto e failure

1.3 Principi del Testing

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

1.4 Attività di Test, Testware e Ruoli del Test

FL-1.4.1 (K2) Riassumere le differenti attività di test e i compiti del test

FL-1.4.2 (K2) Spiegare l'impatto del contesto sul processo di test

FL-1.4.3 (K2) Differenziare il testware che supporta le attività di test

FL-1.4.4 (K2) Spiegare il valore di mantenere la tracciabilità

FL-1.4.5 (K2) Confrontare i differenti ruoli nel testing

1.5 Competenze Fondamentali e Buone Pratiche nel Testing

FL-1.5.1 (K2) Fornire esempi delle competenze generiche richieste per il testing

FL-1.5.2 (K1) Richiamare i vantaggi dell'approccio whole-team

FL-1.5.3 (K2) Distinguere i benefici e gli svantaggi dell'indipendenza del testing

1.1. Cos'è il Testing?

I sistemi software sono parte integrante della nostra vita quotidiana. La maggior parte delle persone ha avuto esperienza con un software che non ha funzionato come previsto. Un software che non funziona correttamente può causare molti problemi, tra cui perdite di denaro, di tempo o di reputazione aziendale e, in casi estremi, persino infortuni o morte. Il testing del software valuta la qualità del software e aiuta a ridurre il rischio di failure nel software in produzione.

Il testing del software è un insieme di attività volte a scoprire i difetti e a valutare la qualità degli artefatti software. Questi artefatti, quando vengono sottoposti al testing, sono conosciuti come oggetti di test. Un'errata percezione comune è che il testing consista solo nell'esecuzione dei test (cioè nell'eseguire il software e verificare i risultati dei test). Il testing del software include anche altre attività e deve essere allineato con il ciclo di vita dello sviluppo software (si veda il capitolo 2).

Un'altra errata percezione comune sul testing è che il testing si focalizzi interamente sulla verifica dell'oggetto del test. Se da un lato il testing comporta la verifica, cioè verificare che il sistema soddisfi i requisiti specificati, dall'altro comporta anche la validazione, cioè verificare che il sistema soddisfi le esigenze degli utenti e degli altri stakeholder nel suo ambiente operativo.

Il testing può essere dinamico o statico. Il testing dinamico comporta l'esecuzione del software, mentre il testing statico non lo richiede. Il testing statico include le review (si veda il capitolo 3) e l'analisi statica. Il testing dinamico utilizza differenti tipi di tecniche di test e approcci del test per derivare i test case (si veda il capitolo 4).

Il testing non è solo un'attività tecnica. Deve anche essere pianificato, gestito, stimato, monitorato e controllato in modo appropriato (si veda il capitolo 5).

I tester utilizzano strumenti (si veda capitolo 6), ma è importante ricordare che il testing è in gran parte un'attività intellettuale, che richiede ai tester conoscenze specialistiche, competenze analitiche e l'applicazione del pensiero critico (critical thinking) e del pensiero sistemico (system thinking) (Myers 2011, Roman 2018).

Lo standard ISO/IEC/IEEE 29119-1 fornisce ulteriori informazioni sui concetti del testing del software.

1.1.1. Obiettivi del Testing

Gli obiettivi tipici del testing sono:

- Valutare i prodotti di lavoro come requisiti, user story, progettazione e codice
- Generare failure e rilevare difetti
- Garantire la copertura richiesta di un oggetto di test
- Ridurre il livello di rischio di una qualità del software inadeguata
- Verificare se i requisiti specificati sono stati soddisfatti
- Verificare che un oggetto di test sia conforme ai requisiti contrattuali, legali e normativi
- Fornire informazioni agli stakeholder per consentire loro di prendere decisioni informate
- Creare fiducia nella qualità dell'oggetto di test
- Validare che l'oggetto di test sia completo e funzioni come richiesto dagli stakeholder

Gli obiettivi del testing possono variare a seconda del contesto, che include il prodotto di lavoro sotto test, il livello di test, i rischi, il ciclo di vita dello sviluppo software (SDLC, Software Development Life Cycle) da applicare, e i fattori legati al contesto di business, ad esempio la struttura organizzativa, le considerazioni sulla concorrenza o il time-to-market.

1.1.2. Testing e Debugging

Il testing e il debugging sono attività separate. Il testing può generare failure causati da difetti nel software (testing dinamico) o può rilevare direttamente difetti nell'oggetto di test (testing statico).

Quando il testing dinamico (si veda il capitolo 4) innesca un failure, il debugging si occupa di trovare le cause di questo failure (difetti), analizzare queste cause ed eliminarle. Il tipico processo di debugging in questo caso prevede:

- Riproduzione di un failure
- Diagnosi (ricerca della root cause)
- Correzione della causa

Il successivo testing confermativo verifica se le correzioni abbiano risolto il problema. Il testing confermativo dovrebbe essere eseguito dalla stessa persona che ha eseguito il test iniziale. Successivamente può essere eseguito anche il regression testing, per verificare se le correzioni apportate stanno causando failure in altre parti dell'oggetto di test (per ulteriori informazioni sul testing confermativo e sul regression testing, si veda il paragrafo 2.2.3).

Quando il testing statico identifica un difetto, il debugging si occupa di eliminarlo. Non è necessaria la riproduzione o la diagnosi, poiché il testing statico rileva direttamente i difetti e non può causare failure (si veda il capitolo 3).

1.2. Perché il Testing è Necessario?

Il testing, come forma di quality control, aiuta a raggiungere gli obiettivi concordati entro l'ambito, i tempi, la qualità e i vincoli di budget stabiliti. Il contributo del testing al successo non dovrebbe essere limitato alle attività del team di test. Tutti gli stakeholder possono utilizzare le loro competenze nel testing per avvicinare il progetto al successo. Il testing dei componenti, dei sistemi e della documentazione associata aiuta a identificare i difetti nel software.

1.2.1. Il Contributo del Testing al Successo

Il testing fornisce un mezzo efficace, dal punto di vista dei costi, per rilevare i difetti. Questi difetti possono poi essere eliminati (attraverso il debugging, un'attività che non riguarda il testing), e quindi il testing contribuisce indirettamente ad ottenere oggetti di test di qualità superiore.

Il testing fornisce un mezzo per valutare direttamente la qualità di un oggetto di test in varie fasi del ciclo di vita dello sviluppo software. Queste misure vengono utilizzate come parte di una più ampia attività di project management, contribuendo a decisioni sul passaggio alla fase successiva del ciclo di vita dello sviluppo software, come la decisione di rilascio.

Il testing fornisce agli utenti una rappresentazione indiretta del progetto di sviluppo. I tester assicurano che la comprensione delle esigenze degli utenti venga presa in considerazione durante tutto il ciclo di vita dello sviluppo software. L'alternativa è di coinvolgere un insieme rappresentativo di utenti nel progetto di sviluppo, cosa che di solito non è possibile a causa dei costi elevati e della mancanza di disponibilità di utenti adeguati.

Il testing può anche essere richiesto per soddisfare requisiti contrattuali o legali, o per conformità verso standard normativi.

1.2.2. Testing e Quality Assurance (QA)

Le persone usano spesso la terminologia "testing" e "Quality Assurance" (Garanzia di Qualità, QA) in modo intercambiabile, ma testing e Quality Assurance non sono la stessa cosa. Il testing è una forma di Quality Control (Controllo della Qualità, QC).

Il Quality Control è un approccio correttivo orientato al prodotto che si focalizza su quelle attività che supportano il raggiungimento di appropriati livelli di qualità. Il testing è una delle principali forme di Quality Control, mentre altre includono i metodi formali (verifica del modello e prova di correttezza), la simulazione e la prototipazione.

La Quality Assurance è un approccio preventivo orientato al processo, che si focalizza sull'implementazione e sul miglioramento dei processi. Si basa sul principio che se un buon processo è seguito in modo corretto, questo genererà un buon prodotto. La Quality Assurance si applica sia al processo di sviluppo sia al processo di test ed è sotto la responsabilità di ogni membro di un progetto.

I risultati del test vengono utilizzati dalla Quality Assurance per fornire un feedback su come bene si stiano eseguendo i processi di sviluppo e di test e, dal Quality Control, per correggere i difetti.

1.2.3. Errori, Difetti, Failure e Root Cause

Gli esseri umani commettono errori, che possono portare all'introduzione di difetti (guasti, bug), che a loro volta possono causare failure. Gli esseri umani commettono errori per ragioni differenti, come la pressione sui tempi, la complessità dei prodotti di lavoro, dei processi, delle infrastrutture o delle interazioni, o semplicemente perché sono stanchi o non hanno una formazione adeguata.

I difetti possono essere rilevati nella documentazione, come una specifica dei requisiti o un test script, nel codice sorgente o in un artefatto di supporto come un file di build. I difetti negli artefatti generati nelle prime fasi del ciclo di vita dello sviluppo software, se non rilevati, spesso portano ad artefatti difettosi nelle fasi successive del ciclo di vita. Se un difetto nel codice viene eseguito, il sistema può fallire, non facendo quello che dovrebbe fare, o facendo qualcosa che non dovrebbe fare, causando un failure. Alcuni difetti causeranno sempre un failure se eseguiti, altri causeranno un failure solo in circostanze specifiche, e altri non causeranno mai un failure.

Gli errori e i difetti non sono l'unica causa di failure. I failure possono essere causati anche da condizioni ambientali, come radiazioni o campi elettromagnetici che causano difetti nel firmware.

Una root cause è il motivo all'origine del verificarsi di un problema (es. una situazione che causa un errore). Le root cause vengono identificate attraverso la root cause analysis, che viene eseguita in genere quando si verifica un failure o viene identificato un difetto. Si ritiene che ulteriori failure o difetti similari possano essere prevenuti, o che la loro frequenza possa essere ridotta, ponendo la giusta attenzione alla root cause, possibilmente eliminandola.

1.3. Principi del Testing

Nel corso degli anni sono stati proposti dei principi del testing che offrono linee guida generali applicabili a tutto il testing. Questo Syllabus descrive sette principi.

1. Il testing mostra la presenza di difetti, ma non la loro assenza. Il testing può mostrare la presenza di difetti nell'oggetto di test, ma non può provarne l'assenza (Buxton 1970). Il testing riduce la probabilità che difetti non rilevati rimangano nell'oggetto di test, ma anche se nessun difetto viene rilevato, il testing non può dimostrare la correttezza dell'oggetto di test, ovvero l'assenza di difetti.

2. Il testing esaustivo è impossibile. Testare tutto non è fattibile tranne che in casi banali (Manna 1978). Piuttosto che tentare di testare in modo esaustivo, le tecniche di test (si veda il capitolo 4), la prioritizzazione dei test case (si veda il paragrafo 5.1.5) e il testing basato sul rischio (si veda il paragrafo 5.2) dovrebbero essere utilizzati per focalizzare l'effort del test.

3. Il testing anticipato permette di risparmiare tempo e denaro. I difetti che vengono eliminati nelle prime fasi del processo non causeranno ulteriori difetti nei prodotti di lavoro derivati. Il costo della qualità sarà ridotto poiché si verificheranno un minor numero di failure nel corso del ciclo di vita dello sviluppo software (Boehm 1981). Per rilevare difetti in anticipo, sia il testing statico (si veda il capitolo 3) che il testing dinamico dovrebbero iniziare il prima possibile (si veda il capitolo 4).

4. I difetti si raggruppano in cluster. Un piccolo numero di componenti del sistema di solito contiene la maggior parte dei difetti rilevati o è responsabile della maggior parte dei failure in esercizio (Enders 1975). Questo fenomeno è una rappresentazione del principio di Pareto. I cluster dei difetti previsti e i cluster dei difetti effettivamente osservati durante il testing o in esercizio sono un input importante per il testing basato sul rischio (si veda il paragrafo 5.2).

5. I test perdono di efficacia. Se gli stessi test vengono ripetuti molte volte, diventano sempre più inefficaci nel rilevare nuovi difetti (Beizer 1990). Per ovviare a questo effetto, i test e i dati di test esistenti potrebbero dover essere modificati, e potrebbe essere necessario scrivere nuovi test. Tuttavia, in alcuni casi, la ripetizione degli stessi test può avere dei benefici, ad esempio nel regression testing automatizzato (si veda il paragrafo 2.2.3).

6. Il testing è dipendente dal contesto. Non esiste un unico approccio universalmente applicabile al testing. Il testing viene eseguito in modo diverso in contesti differenti (Kaner 2011).

7. L'assenza di difetti è un'idea sbagliata. È un'idea sbagliata aspettarsi che la verifica del software garantisca il successo di un sistema. Il testing approfondito di tutti i requisiti specificati e la correzione di tutti i difetti rilevati potrebbe comunque portare al rilascio un sistema che non soddisfa le esigenze e le aspettative degli utenti, che non aiuta a raggiungere gli obiettivi di business del cliente, o che risulta inferiore rispetto ad altri sistemi concorrenti. Oltre alla verifica, dovrebbe essere effettuata anche la validazione (Boehm 1981).

1.4. Attività di Test, Testware e Ruoli del Test

Il testing è dipendente dal contesto ma, ad alto livello, esistono insiemi comuni di attività di test senza le quali è meno probabile che il testing raggiunga i propri obiettivi. Questi insiemi di attività di test costituiscono un processo di test. Il processo di test può essere adattato a una determinata situazione in base a diversi fattori. Le attività di test incluse in questo processo di test, le modalità della loro implementazione e il momento in cui vengono svolte sono normalmente decise come parte della pianificazione dei test per la situazione specifica (si veda il paragrafo 5.1).

I seguenti paragrafi descrivono gli aspetti generali di questo processo di test in termini di attività e compiti del test, impatto del contesto, testware, tracciabilità tra la base di test e il testware, e ruoli del test.

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

1.4.1. Attività e Compiti del Test

Un processo di test è generalmente costituito dai gruppi principali di attività descritti di seguito. Sebbene molte di queste attività possano sembrare seguire una sequenza logica, spesso vengono implementate in modo iterativo o in parallelo. Queste attività di test devono essere adattate (tailoring) al sistema e al progetto.

La **pianificazione dei test** consiste nella definizione degli obiettivi del testing e nella selezione dell'approccio che meglio raggiunga gli obiettivi all'interno dei vincoli imposti dal contesto generale. La pianificazione dei test è spiegata ulteriormente nel paragrafo 5.1.

Monitoraggio e controllo dei test. Il monitoraggio dei test prevede la verifica continua di tutte le attività di test e il confronto dell'avanzamento effettivo rispetto al piano. Il controllo dei test prevede l'implementazione delle azioni necessarie per raggiungere gli obiettivi del testing. Il monitoraggio e il controllo dei test sono ulteriormente spiegati nel paragrafo 5.3.

L'**analisi dei test** include l'analisi della base di test per identificare le funzionalità testabili, e per definire e prioritizzare le condizioni di test associate e dei relativi rischi e livelli di rischio (si veda il paragrafo 5.2). La base di test e gli oggetti di test sono valutati anche per identificare i difetti che questi possono contenere e per valutare la loro testabilità. L'analisi dei test è spesso supportata dall'utilizzo di tecniche di test (si veda il capitolo 4). L'analisi dei test risponde alla domanda "cosa testare?" in termini di criteri di copertura misurabili.

La **progettazione dei test** include l'elaborazione delle condizioni di test in test case e in altro testware (es. test charter). Questa attività prevede spesso l'identificazione degli elementi di copertura, che servono come guida per specificare gli input dei test case. Le tecniche di test (si veda il capitolo 4) possono essere utilizzate per supportare questa attività. La progettazione dei test include anche la definizione dei requisiti dei dati di test, la progettazione dell'ambiente di test e l'identificazione di qualsiasi altra infrastruttura e strumento necessari. La progettazione dei test risponde alla domanda "come testare?".

L'**implementazione dei test** include la creazione o l'acquisizione del testware necessario per l'esecuzione dei test (es. i dati del test). I test case possono essere organizzati in procedure di test e spesso vengono assemblati in test suite. Vengono creati test script manuali e automatizzati. Le procedure di test vengono prioritizzate e organizzate all'interno di una schedulazione di esecuzione dei test per un'esecuzione efficiente dei test (si veda il paragrafo 5.1.5). Viene creato l'ambiente di test, verificando che sia configurato correttamente.

L'**esecuzione dei test** si basa sulla schedulazione di esecuzione dei test (test run). L'esecuzione dei test può essere manuale o automatizzata. L'esecuzione dei test può assumere diverse forme, tra cui il continuous testing o sessioni di pair testing. I risultati effettivi dei test vengono confrontati con i risultati attesi. I risultati dei test vengono registrati (log). Le anomalie vengono analizzate per identificarne le probabili cause. Questa analisi consente di segnalare le anomalie in base ai failure osservati (si veda il paragrafo 5.5).

Le attività di **completamento dei test** di solito si svolgono in corrispondenza delle milestone di progetto (es. rilascio, fine dell'iterazione, completamento del livello di test) per qualsiasi difetto non risolto, per le change request o per gli item creati nel product backlog. Tutto il testware che può essere utile in futuro

viene identificato e archiviato, o consegnato ai team appropriati. L'ambiente di test viene finalizzato in uno stato concordato. Le attività di test vengono analizzate per identificare le lessons learned (lezioni apprese) e i miglioramenti per le iterazioni, le release o i progetti futuri (si veda il paragrafo 2.1.6). Un test completion report viene creato e comunicato agli stakeholder.

1.4.2. Il Processo di Test nel Contesto

Il testing non viene svolto in modo isolato. Le attività di test sono parte integrante dei processi di sviluppo eseguiti all'interno di un'organizzazione. Il testing è finanziato anche dagli stakeholder e il loro obiettivo finale è quello di contribuire a soddisfare le esigenze di business degli stakeholder. Il modo in cui il testing viene eseguito dipenderà quindi da una serie di fattori contestuali, che includono:

- Stakeholder (esigenze, aspettative, requisiti, disponibilità a collaborare, ecc.)
- Membri del team (competenze, conoscenze, livello di esperienza, disponibilità, esigenze di formazione, ecc.)
- Dominio di business (criticità dell'oggetto di test, rischi identificati, esigenze di mercato, normative legali specifiche, ecc.)
- Fattori tecnici (tipo di software, architettura del prodotto, tecnologia utilizzata, ecc.)
- Vincoli di progetto (ambito, tempo, budget, risorse, ecc.)
- Fattori organizzativi (struttura organizzativa, politiche esistenti, pratiche utilizzate, ecc.)
- Ciclo di vita dello sviluppo software (pratiche di ingegneria del software, metodi di sviluppo, ecc.)
- Strumenti (disponibilità, usabilità, conformità, ecc.)

Questi fattori avranno un impatto su molte questioni relative al testing, che includono: strategia di test, tecniche di test utilizzate, grado di test automation, livello di copertura richiesto, livello di dettaglio della documentazione di test, reporting, ecc.

1.4.3. Testware

Il testware viene creato come prodotto di lavoro in output delle attività di test descritte nel paragrafo 1.4.1. Il modo in cui le diverse organizzazioni producono, modellano, nominano, organizzano e gestiscono i loro prodotti di lavoro varia in modo significativo. Un configuration management appropriato (si veda il paragrafo 5.4) assicura la consistenza e l'integrità dei prodotti di lavoro. La seguente lista dei prodotti di lavoro non è esaustiva:

- **I prodotti di lavoro della pianificazione dei test** includono: test plan, schedulazione dei test, risk register, criteri di ingresso e criteri di uscita (si veda il paragrafo 5.1). Il risk register (registro dei rischi) è una lista dei rischi identificati con la relativa probabilità di rischio, l'impatto del rischio e le informazioni sulla mitigazione del rischio (si veda il paragrafo 5.2). La schedulazione dei test, il risk register, i criteri di ingresso e i criteri di uscita sono spesso parte del test plan.
- **I prodotti di lavoro del monitoraggio e controllo dei test** includono: test progress report (si veda il paragrafo 5.3.2), documentazione delle direttive di controllo (si veda il paragrafo 5.3) e informazioni sui rischi (si veda il paragrafo 5.2).

- **I prodotti di lavoro dell'analisi dei test** includono: condizioni di test (prioritizzate) (es. criteri di accettazione, si veda il paragrafo 4.5.2), e defect report sui difetti rilevati nella base di test (se non corretti direttamente).
- **I prodotti di lavoro della progettazione dei test** includono: test case (prioritizzati), test charter, elementi di copertura, requisiti dei dati di test e requisiti dell'ambiente di test.
- **I prodotti di lavoro dell'implementazione dei test** includono: procedure di test, test script automatizzati, test suite, dati di test, schedulazione dell'esecuzione dei test, ed elementi dell'ambiente di test. Esempi di elementi dell'ambiente di test includono: stub, driver, simulatori e virtualizzazioni di servizi.
- **I prodotti di lavoro dell'esecuzione dei test** includono: test log e defect report (si veda il paragrafo 5.5).
- **I prodotti di lavoro per il completamento dei test** includono: test completion report (si veda il paragrafo 5.3.2), action item per il miglioramento dei progetti o delle iterazioni successive, lessons learned documentate e change request (es. come item del product backlog).

1.4.4. Tracciabilità tra la Base di Test e il Testware

Al fine di implementare un efficace monitoraggio e controllo dei test, è importante stabilire e mantenere durante tutto il processo di test la tracciabilità tra gli elementi della base di test, il testware associato a questi elementi (es. condizioni di test, rischi, test case), i risultati dei test e i difetti rilevati.

Una tracciabilità accurata supporta la valutazione della copertura, quindi è molto utile se i criteri di copertura misurabili sono definiti nella base di test. I criteri di copertura possono funzionare come Key Performance Indicator (KPI) per guidare le attività che mostrano in che misura gli obiettivi del testing sono stati raggiunti (si veda il paragrafo 1.1.1). Ad esempio:

- La tracciabilità dei test case verso i requisiti può verificare che i requisiti siano coperti dai test case.
- La tracciabilità dei risultati dei test verso i rischi può essere utilizzata per valutare il livello di rischio residuo di un oggetto di test.

Oltre a valutare la copertura, una buona tracciabilità consente di determinare l'impatto delle modifiche, facilita l'auditing del testing e aiuta a soddisfare i criteri di governance IT. Una buona tracciabilità rende anche più facilmente comprensibili i test progress report e i test completion report, includendo lo stato degli elementi della base di test. Questo può anche aiutare a comunicare gli aspetti tecnici del testing agli stakeholder in modo comprensibile. La tracciabilità fornisce informazioni per valutare la qualità del prodotto, la capacità del processo e l'avanzamento del progetto rispetto agli obiettivi di business.

1.4.5. Ruoli nel Testing

In questo Syllabus, vengono coperti due ruoli principali nel testing: un ruolo di test management e un ruolo di testing. Le attività e i compiti assegnati a questi due ruoli dipendono da diversi fattori, come il contesto di progetto e di prodotto, le competenze delle persone che ricoprono questi ruoli, e l'organizzazione.

Il ruolo di test management assume la responsabilità generale del processo di test, del team di test e della leadership delle attività di test. Il ruolo di test management si focalizza principalmente sulle attività di pianificazione dei test, di monitoraggio e controllo dei test, e di completamento dei test. Il modo in cui il ruolo di test management viene svolto varia a seconda del contesto. Ad esempio, nello sviluppo software

Agile, alcuni dei task di test management possono essere gestiti dal team Agile. I task che riguardano più team o l'intera organizzazione possono essere svolti da test manager esterni al team di sviluppo.

Il ruolo di testing assume la responsabilità generale dell'aspetto ingegneristico (tecnico) del testing. Il ruolo di testing si focalizza principalmente sulle attività di analisi dei test, di progettazione dei test, di implementazione dei test e di esecuzione dei test.

Questi ruoli possono essere coperti da persone differenti in momenti differenti. Ad esempio, il ruolo di test management può essere svolto da un team leader, da un test manager, da un responsabile dello sviluppo, ecc. È anche possibile che una persona assuma contemporaneamente i ruoli di testing e di test management.

1.5. Competenze Fondamentali e Buone Pratiche nel Testing

La competenza è l'abilità di fare bene qualcosa che deriva dalla conoscenza, dalla pratica e dall'attitudine. Un buon tester dovrebbe possedere alcune competenze fondamentali per svolgere bene il proprio lavoro. Buoni tester dovrebbero essere efficaci giocatori di squadra e dovrebbero essere in grado di eseguire il testing a differenti livelli di indipendenza.

1.5.1. Competenze Generiche Richieste per il Testing

Anche se generiche, le seguenti competenze sono particolarmente rilevanti per i tester:

- Conoscenza del testing (per aumentare l'efficacia del testing, ad esempio utilizzando le tecniche di test)
- Accuratezza, cautela, curiosità, attenzione ai dettagli, metodicità (per identificare i difetti, soprattutto quelli difficili da rilevare)
- Buone competenze di comunicazione, di active listening (ascolto attivo), di gioco di squadra (team player, per interagire efficacemente con tutti gli stakeholder, per trasmettere informazioni agli altri, per farsi capire, per segnalare e discutere i difetti)
- Pensiero analitico, pensiero critico, creatività (per aumentare l'efficacia del testing)
- Conoscenze tecniche (per aumentare l'efficienza del testing, per esempio utilizzando strumenti di test appropriati)
- Conoscenza del dominio (per essere in grado di comprendere e comunicare con gli utenti finali/rappresentanti di business)

I tester sono spesso portatori di cattive notizie. È un tratto umano comune quello di incolpare il portatore di cattive notizie. Questo rende fondamentali le competenze di comunicazione dei tester. La comunicazione dei risultati dei test può essere percepita come una critica al prodotto e al suo autore. Il confirmation bias (pregiudizio di conferma) può rendere difficile accettare informazioni che sono in disaccordo con le convinzioni e le idee ritenute valide. Alcune persone possono percepire il testing come un'attività distruttiva, anche se contribuisce notevolmente al successo del progetto e alla qualità del prodotto. Per cercare di migliorare questa percezione, le informazioni sui difetti e sui failure dovrebbero essere comunicate in modo costruttivo.

1.5.2. Approccio Whole-Team

Una delle competenze importanti per un tester è la capacità di lavorare efficacemente in un contesto di team e di contribuire positivamente agli obiettivi del team. L'approccio whole-team, una pratica proveniente dall'Extreme Programming (si veda il paragrafo 2.1), si basa su questa competenza.

Nell'approccio whole-team, ogni membro del team con le conoscenze e le competenze necessarie può svolgere qualsiasi compito, e tutti sono responsabili della qualità. I membri del team condividono lo stesso spazio di lavoro (fisico o virtuale), poiché la co-locazione facilita la comunicazione e l'interazione. L'approccio whole-team migliora le dinamiche del team, aumenta la comunicazione e la collaborazione all'interno del team, e crea sinergie consentendo di utilizzare le diverse competenze all'interno del team a vantaggio del progetto.

I tester lavorano a stretto contatto con gli altri membri del team per garantire che vengano raggiunti i livelli di qualità desiderati. Questo include la collaborazione con i rappresentanti di business per aiutarli a creare test di accettazione adeguati e la collaborazione con gli sviluppatori per concordare la strategia di test e decidere gli approcci di test automation. I tester possono quindi trasferire le conoscenze del testing ad altri membri del team e influenzare lo sviluppo del prodotto.

A seconda del contesto, l'approccio whole-team potrebbe non essere sempre appropriato. Ad esempio, in alcune situazioni, come un progetto safety-critical, può essere richiesto un alto livello di indipendenza del testing.

1.5.3. Indipendenza del Testing

Un certo grado di indipendenza rende il tester più efficace nel rilevare i difetti, grazie alle differenze tra i cognitive bias (pregiudizi cognitivi) dell'autore e del tester (Salman 1995). L'indipendenza, tuttavia, non sostituisce la familiarità: ad esempio gli sviluppatori possono trovare efficacemente molti difetti nel proprio codice.

I prodotti di lavoro possono essere testati dal loro autore (nessuna indipendenza), dai colleghi dell'autore dello stesso team (poca indipendenza), da tester esterni al team dell'autore ma all'interno dell'organizzazione (alta indipendenza) o da tester esterni all'organizzazione (indipendenza molto alta). Per la maggior parte dei progetti, di solito è meglio svolgere il testing con livelli multipli di indipendenza (es. gli sviluppatori eseguono il testing di componente e il testing di integrazione dei componenti, il team di test esegue il testing di sistema e il testing di integrazione dei sistemi, e i rappresentanti di business eseguono il testing di accettazione).

Il principale vantaggio dell'indipendenza del testing è che i tester indipendenti sono in grado di riconoscere differenti tipi di failure e difetti rispetto agli sviluppatori grazie ai loro diversi background, alle loro specifiche prospettive tecniche e bias (pregiudizi). Inoltre, un tester indipendente può verificare, sfidare o confutare le ipotesi formulate dagli stakeholder durante la specifica e l'implementazione del sistema.

Tuttavia, ci sono anche alcuni svantaggi. I tester indipendenti possono essere isolati dal team di sviluppo, e questo può portare a una mancanza di collaborazione, a problemi di comunicazione o a un rapporto conflittuale con il team di sviluppo. Gli sviluppatori possono perdere il senso di responsabilità per la qualità. I tester indipendenti possono essere visti come un collo di bottiglia o essere incolpati per i ritardi nel rilascio.

2. Il Testing all'Interno del Ciclo di Vita dello Sviluppo Software - 130 minuti

Parole chiave

livello di test, oggetto di test, regression testing, shift-left, testing black-box, testing confermativo, testing di accettazione, testing di componente, testing di integrazione, testing di integrazione dei componenti, testing di integrazione dei sistemi, testing di manutenzione, testing di sistema, testing funzionale, testing non funzionale, testing white-box, tipo di test

Obiettivi di Apprendimento per il Capitolo 2:

2.1 Il Testing nel Contesto di un Ciclo di Vita dello Sviluppo Software

FL-2.1.1 (K2) Spiegare l'impatto sul testing del ciclo di vita dello sviluppo software selezionato

FL-2.1.2 (K1) Ricordare le buone pratiche del testing che si applicano a tutti i cicli di vita dello sviluppo software

FL-2.1.3 (K1) Ricordare gli esempi di approcci allo sviluppo test-first

FL-2.1.4 (K2) Riassumere come DevOps potrebbe avere un impatto sul testing

FL-2.1.5 (K2) Spiegare l'approccio shift-left

FL-2.1.6 (K2) Spiegare come le retrospettive possono essere utilizzate come meccanismo per il process improvement (miglioramento del processo)

2.2 Livelli di Test e Tipi di Test

FL-2.2.1 (K2) Distinguere i differenti livelli di test

FL-2.2.2 (K2) Distinguere i differenti tipi di test

FL-2.2.3 (K2) Distinguere il testing confermativo dal regression testing

2.3 Testing di Manutenzione

FL-2.3.1 (K2) Riassumere il testing di manutenzione e i suoi trigger di attivazione

2.1. Il Testing nel Contesto di un Ciclo di Vita dello Sviluppo Software

Un ciclo di vita dello sviluppo software (SDLC, Software Development LifeCycle) è una rappresentazione astratta e di alto livello del processo di sviluppo software. Un modello di SDLC definisce come le differenti fasi di sviluppo e i tipi di attività svolte all'interno di questo processo si relazionano tra loro, sia logicamente che cronologicamente. Esempi di modelli di SDLC includono: modelli di sviluppo sequenziale (es. waterfall, V-model), modelli di sviluppo iterativo (es. modello a spirale, prototipazione) e modelli di sviluppo incrementale (es. Unified Process).

Alcune attività all'interno dei processi di sviluppo software possono anche essere descritte da metodi di sviluppo software più dettagliati e da pratiche Agile. Esempi includono: acceptance test-driven development (ATDD), behavior-driven development (BDD), domain-driven design (DDD), extreme programming (XP), feature-driven development (FDD), Kanban, Lean IT, Scrum, e test-driven development (TDD).

2.1.1. Impatto del Ciclo di Vita dello Sviluppo Software sul Testing

Per avere successo, il testing deve essere adattato al SDLC. La scelta del SDLC ha un impatto su:

- Ambito e tempistica delle attività di test (es. livelli di test e tipi di test)
- Livello di dettaglio della documentazione di test
- Scelta delle tecniche di test e dell'approccio del test
- Estensione della test automation
- Ruolo e responsabilità di un tester

Nei modelli di sviluppo sequenziale, nelle fasi iniziali i tester partecipano normalmente alle review dei requisiti, all'analisi dei test e alla progettazione dei test. Il codice eseguibile viene generalmente creato nelle fasi successive, per cui, in genere, il testing dinamico non può essere eseguito nelle prime fasi del SDLC.

In alcuni modelli di sviluppo iterativo e incrementale, si assume che ogni iterazione rilasci un prototipo o un incremento di prodotto funzionante. Questo implica che in ogni iterazione può essere eseguito sia testing statico, sia testing dinamico a tutti i livelli di test. Il rilascio frequente di incrementi richiede un feedback rapido e un ampio regression testing.

Lo sviluppo software Agile assume che le modifiche possano verificarsi nel corso del progetto. Pertanto, nei progetti Agile è preferibile una documentazione leggera (“lightweight”) del prodotto di lavoro e un ampia test automation per facilitare il regression testing. Inoltre, la maggior parte del testing manuale tende a essere eseguito utilizzando tecniche di test basate sull'esperienza (si veda il paragrafo 4.4) che non richiedono un'analisi e una progettazione approfondita dei test.

2.1.2. Ciclo di Vita dello Sviluppo Software e Buone Pratiche del Testing

Le buone pratiche del testing, indipendentemente dal modello del SDLC scelto, includono quanto segue:

- Per ogni attività di sviluppo software, esiste una corrispondente attività di test, in modo che tutte le attività di sviluppo siano soggette al controllo di qualità
- I diversi livelli di test (si veda il paragrafo 2.2.1) hanno obiettivi del testing specifici e differenti, che consentono di eseguire un testing adeguatamente completo evitando la ridondanza

- L'analisi e la progettazione dei test per un determinato livello di test inizia durante la corrispondente fase di sviluppo del SDLC, in modo che il testing possa aderire al principio del testing anticipato (si veda il paragrafo 1.3)
- I tester sono coinvolti nella review dei prodotti di lavoro non appena sono disponibili i draft di questa documentazione, in modo che il testing anticipato e la rilevazione dei difetti possano supportare la strategia shift-left (si veda il paragrafo 2.1.5).

2.1.3. Il Testing come Driver per lo Sviluppo Software

TDD, ATDD e BDD sono approcci di sviluppo simili, in cui i test sono definiti come mezzi per guidare lo sviluppo. Ognuno di questi approcci implementa il principio del testing anticipato (si veda il paragrafo 1.3) e segue un approccio shift-left (si veda il paragrafo 2.1.5), poiché i test vengono definiti prima di aver scritto il codice. Questi approcci supportano un modello di sviluppo iterativo e sono caratterizzati come segue:

Test-Driven Development (TDD):

- Guida la codifica attraverso i test case (invece che tramite una progettazione completa del software) (Beck 2003)
- Prima vengono scritti i test, poi viene scritto il codice per soddisfare i test, e infine viene eseguito il refactoring dei test e del codice

Acceptance Test-Driven Development (ATDD) (si veda il paragrafo 4.5.3):

- Deriva i test dai criteri di accettazione come parte del processo di progettazione del sistema (Gärtner 2011)
- I test vengono scritti prima che venga sviluppata la parte dell'applicazione necessaria per soddisfare i test

Behavior-Driven Development (BDD):

- Esprime il comportamento desiderato di un'applicazione con test case scritti in una forma semplice del linguaggio naturale, facile da comprendere per gli stakeholder - di solito utilizzando il formato Given/When/Then (Dato/Quando/Allora). (Chelimsky 2010)
- I test case vengono quindi tradotti automaticamente in test eseguibili

Per tutti gli approcci descritti sopra, i test possono esistere manutenuti come automatizzati per garantire la qualità del codice in caso di futuri adeguamenti/refactoring.

2.1.4. DevOps e Testing

DevOps è un approccio organizzativo che mira a creare sinergie facendo sì che lo sviluppo (incluso il testing) e le operations lavorino insieme per raggiungere un insieme di obiettivi comuni. DevOps richiede un cambiamento culturale all'interno di un'organizzazione per colmare il divario tra lo sviluppo (incluso il testing) e le operations, attribuendo alle rispettive funzioni uguale valore. DevOps promuove l'autonomia del team, il feedback rapido, le toolchain integrate e le pratiche tecniche come la Continuous Integration (CI) e la Continuous Delivery (CD). Questo permette ai team di costruire, testare e rilasciare codice di alta qualità più velocemente, attraverso una pipeline di delivery (rilascio) DevOps (Kim 2016).

Dal punto di vista del testing, alcuni dei vantaggi di DevOps sono:

- Feedback rapido sulla qualità del codice e sull'eventuale impatto negativo delle modifiche al codice esistente
- La Continuous Integration (CI) promuove un approccio shift-left nel testing (si veda il paragrafo 2.1.5), incoraggiando gli sviluppatori a rilasciare codice di alta qualità insieme ai test di componente e a svolgere analisi statica
- Promuove processi automatizzati come CI/CD che facilitano la creazione di ambienti di test stabili
- Aumenta la visuale sulle caratteristiche qualitative non funzionali (es. prestazioni, affidabilità)
- L'automazione attraverso una pipeline di delivery riduce la necessità di eseguire testing manuale ripetitivo
- Il rischio di regressione è minimizzato grazie all'ampiezza e alla gamma dei regression test automatizzati.

DevOps non è esente da rischi e sfide, che includono:

- La pipeline di delivery DevOps deve essere definita e predisposta
- Gli strumenti di CI/CD devono essere introdotti e manutenuti
- La test automation richiede risorse aggiuntive e può essere difficile da predisporre e manutenere

Sebbene DevOps preveda un alto livello di testing automatizzato, il testing manuale, soprattutto dal punto di vista dell'utente, sarà comunque ancora necessario.

2.1.5. Approccio Shift-Left

Il principio del testing anticipato (si veda il paragrafo 1.3) viene a volte indicato come shift-left perché si tratta di un approccio dove il testing viene eseguito negli stadi iniziali del ciclo di vita dello sviluppo software. L'approccio shift-left normalmente suggerisce che il testing dovrebbe essere eseguito prima (es. senza aspettare che il codice sia implementato o che i componenti siano integrati), ma ciò non significa che il testing successivo nel SDLC debba essere trascurato.

Esistono alcune buone pratiche che illustrano come ottenere uno "shift-left" nel testing, che includono:

- Eseguire la review delle specifiche dalla prospettiva del testing. Queste attività di review delle specifiche spesso rilevano potenziali difetti, come ambiguità, incompletezze e inconsistenze
- Scrivere test case prima della stesura del codice e far eseguire il codice in una test harness durante l'implementazione del codice
- Utilizzare la Continuous Integration, e ancora meglio la Continuous Delivery (CD), poiché ciò permette di ottenere un feedback rapido e di eseguire testing di componente automatizzato quando il codice sorgente viene rilasciato nel repository del codice
- Completare l'analisi statica del codice sorgente prima del testing dinamico o come parte di un processo automatizzato
- Eseguire testing non funzionale a partire dal livello di test di componente, quando possibile. Questa è una forma di shift-left, poiché questi tipi di test non funzionali tendono a essere eseguiti più tardi nel SDLC, quando sono disponibili un sistema completo e un ambiente di test rappresentativo.

Un approccio shift-left potrebbe comportare attività di formazione, effort e/o costi aggiuntivi nelle prime fasi del processo, ma si prevede un risparmio di effort e/o costi nelle fasi successive.

Per l'approccio shift-left è importante che gli stakeholder siano convinti e credano in questo concetto.

2.1.6. Retrospettive e Process Improvement (Miglioramento del Processo)

Le retrospettive (note anche come "meeting post-progetto" e “retrospettive di progetto”) si svolgono spesso alla fine di un progetto o di un'iterazione, in corrispondenza di una milestone di rilascio, oppure possono essere organizzate quando necessario. Le tempistiche e l'organizzazione delle retrospettive dipendono dal particolare modello del SDLC che viene seguito. Durante questi meeting i partecipanti (non solo i tester, ma anche, ad esempio, sviluppatori, architetti, product owner, business analyst) discutono su:

- Cosa ha avuto successo e dovrebbe essere mantenuto?
- Cosa non ha avuto successo e potrebbe essere migliorato?
- Come incorporare i miglioramenti e mantenere i successi in futuro?

I risultati dovrebbero essere memorizzati e di solito fanno parte del test completion report (si veda il paragrafo 5.3.2). Le retrospettive sono fondamentali per il successo dell'implementazione del continuous improvement (miglioramento continuo), ed è importante che a tutti i miglioramenti raccomandati venga dato l’opportuno seguito.

I vantaggi tipici per il testing includono:

- Aumento dell'efficacia e dell'efficienza dei test (es. implementando suggerimenti per il process improvement)
- Aumento della qualità del testware (es. attraverso la review congiunta dei processi di test)
- Coesione e apprendimento del team (es. grazie all'opportunità di evidenziare problemi e proporre punti di miglioramento)
- Miglioramento della qualità della base di test (es., poiché le carenze nell'estensione e nella qualità dei requisiti possono essere affrontate e risolte).
- Migliore cooperazione tra sviluppo e testing (es. poiché la collaborazione viene rivista e ottimizzata regolarmente).

2.2. Livelli di Test e Tipi di Test

I livelli di test sono gruppi di attività di test che vengono organizzate e gestite insieme. Ogni livello di test è un'istanza del processo di test, eseguita in relazione al software in una determinata fase di sviluppo, dai singoli componenti ai sistemi completi o, dove applicabile, ai sistemi di sistemi.

I livelli di test sono correlati ad altre attività all'interno del SDLC. Nei modelli del SDLC sequenziali, i livelli di test sono spesso definiti in modo che i criteri di uscita di un livello siano parte dei criteri di ingresso del livello successivo. In alcuni modelli iterativi, questo non può essere applicato. Le attività di sviluppo possono estendersi a più livelli di test. I livelli di test possono sovrapporsi nel tempo.

I tipi di test sono gruppi di attività di test relative a specifiche caratteristiche di qualità e la maggior parte di queste attività di test può essere eseguita a ogni livello di test.

2.2.1. Livelli di Test

In questo Syllabus sono descritti i seguenti cinque livelli di test:

- Il **testing di componente** (conosciuto anche come unit testing) si focalizza sul testing di un componente in modo isolato. Spesso richiede un supporto specifico, come una test harness o un framework di unit test. Il testing di componente viene normalmente eseguito dagli sviluppatori nei loro ambienti di sviluppo.
- Il **testing di integrazione dei componenti** (noto anche come unit integration testing) si focalizza sul testing delle interfacce e delle interazioni tra componenti. Il testing di integrazione dei componenti dipende fortemente dagli approcci seguiti per la strategia di integrazione, come bottom-up, top-down o big-bang.
- Il **testing di sistema** si focalizza sul comportamento e sulle capacità complessive di un sistema o prodotto completo, spesso includendo il testing funzionale end-to-end e il testing non funzionale delle caratteristiche di qualità. Per alcune caratteristiche di qualità non funzionali, è preferibile eseguire il testing di un sistema completo in un ambiente di test rappresentativo (es. l'usabilità). È possibile anche utilizzare simulazioni di sottosistemi. Il testing di sistema può essere eseguito da un team di test indipendente ed è relativo alle specifiche del sistema.
- Il **testing di integrazione dei sistemi** si focalizza sul testing delle interfacce del sistema sotto test con altri sistemi e servizi esterni. Il testing di integrazione dei sistemi richiede ambienti di test adeguati, preferibilmente simili all'ambiente operativo.
- Il **testing di accettazione** si focalizza sul testing e sul dimostrare la readiness per il rilascio, e questo significa che il sistema soddisfa le esigenze di business dell'utente. Idealmente, il testing di accettazione dovrebbe essere eseguito dagli utenti previsti. Le principali forme di testing di accettazione sono: user acceptance testing (UAT, testing di accettazione utente), operational acceptance testing (testing di accettazione operativa), testing di accettazione contrattuale e di normativa, alpha testing e beta testing.

I livelli di test si distinguono secondo il seguente elenco, non esaustivo, di attributi, per evitare la sovrapposizione delle attività di test:

- Oggetto di test
- Obiettivi del testing
- Base di test
- Difetti e failure
- Approccio e responsabilità.

2.2.2. Tipi di test

Esistono molti tipi di test che possono essere applicati nei progetti. In questo Syllabus vengono affrontati i seguenti quattro tipi di test:

Il **testing funzionale** valuta le funzioni che un componente o sistema dovrebbe svolgere. Le funzioni rappresentano "cosa" l'oggetto di test dovrebbe fare. L'obiettivo principale del testing funzionale è verificare la completezza funzionale, la correttezza funzionale e l'appropriatezza funzionale.

Il **testing non funzionale** valuta attributi diversi dalle caratteristiche funzionali di un componente o di un sistema. Il testing non funzionale è la verifica di "quanto bene si comporta il sistema". L'obiettivo principale del testing non funzionale è verificare le caratteristiche di qualità non funzionali del software. Lo standard ISO/IEC 25010 fornisce la seguente classificazione delle caratteristiche di qualità non funzionali del software:

- Efficienza delle prestazioni
- Compatibilità
- Usabilità
- Affidabilità
- Sicurezza
- Manutenibilità
- Portabilità

A volte è appropriato che il testing non funzionale inizi negli stadi iniziali del ciclo di vita (es. come parte delle review, del testing di componente o del testing di sistema). Molti test non funzionali derivano dai test funzionali, in quanto utilizzano gli stessi test funzionali, ma verificano che durante l'esecuzione della funzione sia soddisfatto un vincolo non funzionale (es. verificando che una funzione venga eseguita entro un tempo specifico o che una funzione possa essere trasferita su una nuova piattaforma). La scoperta tardiva di difetti non funzionali può rappresentare una seria minaccia per il successo di un progetto. Il testing non funzionale richiede talvolta un ambiente di test molto specifico, come un laboratorio di usabilità per il testing di usabilità.

Il **testing black-box** (si veda il paragrafo 4.2) è basato sulle specifiche e deriva i test dalla documentazione esterna all'oggetto di test. L'obiettivo principale del testing black-box è verificare il comportamento del sistema rispetto alle sue specifiche.

Il **testing white-box** (si veda il paragrafo 4.3) è basato sulla struttura e deriva i test dall'implementazione o dalla struttura interna del sistema (es. codice, architettura, workflow e flussi dati). L'obiettivo principale del testing white-box è coprire la struttura sottostante con i test a un livello di copertura accettabile.

Tutti e quattro i tipi di test sopra menzionati possono essere applicati a tutti i livelli di test, anche se il focus sarà diverso ad ogni livello. È possibile utilizzare differenti tecniche di test per derivare le condizioni di test e i test case per tutti i tipi di test descritti.

2.2.3. Testing Confermativo e Regression Testing

Le modifiche vengono normalmente apportate a un componente o a un sistema per migliorarlo con l'inserimento di una nuova funzionalità o per correggerlo eliminando un difetto. Il testing dovrebbe quindi includere anche il testing confermativo e il regression testing.

Il **testing confermativo** verifica e conferma che un difetto originale è stato risolto con successo. In base al rischio, la versione del software che è stata corretta può essere testata in modi differenti, che includono:

- eseguire tutti i test case che in precedenza erano falliti a causa del difetto o, anche

- aggiungere nuovi test case per coprire le modifiche che sono state necessarie per correggere il difetto.

Tuttavia, quando il tempo o il denaro a disposizione per la correzione dei difetti non sono sufficienti, il testing confermativo potrebbe essere limitato a esercitare semplicemente i passi che dovrebbero riprodurre il failure causato dal difetto, e a verificare che il failure non si verifichi.

Il **regression testing** conferma che una modifica, inclusa la correzione di un difetto che è già stata verificata con il testing confermativo, non abbia causato conseguenze negative. Queste conseguenze negative potrebbero riguardare lo stesso componente in cui è stata apportata la modifica, altri componenti dello stesso sistema o addirittura altri sistemi collegati. Il regression testing può non essere limitato all'oggetto di test stesso, ma può anche riguardare l'ambiente. È consigliabile eseguire prima un'analisi degli impatti (change impact analysis) per ottimizzare l'estensione del regression testing. L'analisi degli impatti mostra quali parti del software potrebbero essere interessate dalla modifica.

Le suite di regression test vengono eseguite molte volte e in genere il numero di regression test aumenterà ad ogni iterazione o rilascio; quindi, il regression testing è un forte candidato per l'automazione. L'automazione di questi test dovrebbe iniziare fin dalle prime fasi del progetto. Quando si utilizza la Continuous Integration, come nel DevOps (si veda il paragrafo 2.1.4), è buona pratica includere anche i regression test automatizzati. In base alla situazione, questo può includere i regression test a differenti livelli.

Il testing confermativo e/o il regression testing per l'oggetto di test sono necessari per tutti i livelli di test, se i difetti vengono corretti e/o se vengono apportate modifiche a questi livelli di test.

2.3. Testing di Manutenzione

Esistono diverse categorie di manutenzione: può essere correttiva, adattativa alle modifiche nell'ambiente, o migliorativa per migliorare le prestazioni o la manutenibilità (per i dettagli, si veda ISO/IEC 14764). La manutenzione può comportare rilasci pianificati e rilasci non pianificati (hot fix). L'analisi degli impatti può essere svolta prima di apportare una modifica, per aiutare a decidere se la modifica stessa debba essere eseguita in base alle potenziali conseguenze in altre aree del sistema. Il testing delle modifiche apportate a un sistema in produzione include sia la valutazione del successo dell'implementazione della modifica, sia la verifica di eventuali regressioni nelle parti del sistema rimaste invariate (che sono di solito la maggior parte del sistema).

L'ambito dei testing di manutenzione dipende tipicamente da:

- Il livello di rischio della modifica
- Le dimensioni del sistema esistente
- La dimensione della modifica

I trigger di attivazione della manutenzione e del testing di manutenzione possono essere classificati come segue:

- Modifiche, come miglioramenti pianificati (cioè release-based), modifiche correttive oppure hot fix
- Aggiornamenti o migrazioni dell'ambiente operativo, ad esempio da una piattaforma a un'altra, che possono richiedere test associati al nuovo ambiente e al software modificato, oppure test di conversione dei dati quando i dati vengono migrati da un'altra applicazione al sistema sottoposto a manutenzione

- Ritiro, ad esempio quando un'applicazione raggiunge la fine del suo ciclo di vita. Se un sistema viene ritirato, può essere necessario eseguire il testing dell'archiviazione dei dati quando sono richiesti lunghi periodi di data-retention (conservazione dei dati). Può anche essere necessario il testing delle procedure di retrieve (recupero) e restore (ripristino) dei dati dopo l'archiviazione, nel caso in cui alcuni dati siano necessari durante il periodo di archiviazione.

3. Testing Statico - 80 minuti

Parole Chiave

anomalia, analisi statica, ispezione, review, review formale, review informale, review tecnica, testing dinamico, testing statico, walkthrough

Obiettivi di Apprendimento per il Capitolo 3:

3.1 Fondamenti del Testing Statico

FL-3.1.1 (K1) Riconoscere i tipi di prodotti che possono essere esaminati con le diverse tecniche di test statico

FL-3.1.2 (K2) Spiegare il valore del testing statico

FL-3.1.3 (K2) Confrontare e mettere in contrapposizione il testing statico e dinamico

3.2 Feedback e Processo di Review

FL-3.2.1 (K1) Identificare i benefici di un feedback anticipato e frequente da parte degli stakeholder

FL-3.2.2 (K2) Riassumere le attività del processo di review

FL-3.2.3 (K1) Ricordare le responsabilità assegnate ai ruoli principali durante l'esecuzione delle review

FL-3.2.4 (K2) Confrontare e mettere in contrapposizione i diversi tipi di review

FL-3.2.5 (K1) Ricordare i fattori che contribuiscono al successo di una review

3.1. Fondamenti del Testing Statico

A differenza del testing dinamico, nel testing statico il software sotto test non deve essere eseguito. Il codice, le specifiche di processo, le specifiche dell'architettura di sistema o altri prodotti di lavoro vengono valutati attraverso l'esame manuale (es. la review) o con l'aiuto di uno strumento (es. l'analisi statica). Gli obiettivi del testing includono il miglioramento della qualità, la rilevazione dei difetti e la valutazione di caratteristiche quali leggibilità, completezza, correttezza, testabilità e consistenza. Il testing statico può essere applicato sia per la verifica sia per la validazione.

I tester, i rappresentanti di business e gli sviluppatori lavorano insieme durante il mapping degli esempi, la scrittura collaborativa delle user story e le sessioni di raffinamento del backlog, per garantire che le user story e i relativi prodotti di lavoro soddisfino i criteri definiti, per esempio la Definition of Ready (si veda il paragrafo 5.1.3). Le tecniche di review possono essere applicate per assicurare che le user story siano complete e comprensibili e che includano criteri di accettazione testabili. Ponendo le domande giuste, i tester esplorano, mettono in discussione e contribuiscono a migliorare le user story proposte.

L'analisi statica può identificare i problemi prima del testing dinamico e spesso richiede un effort minore, poiché non sono richiesti test case e vengono generalmente utilizzati degli strumenti (si veda il capitolo 6). L'analisi statica è spesso incorporata nei framework di Continuous Integration (si veda il paragrafo 2.1.4). Sebbene sia largamente utilizzata per rilevare difetti specifici del codice, l'analisi statica viene utilizzata anche per valutare la manutenibilità e la sicurezza. Gli strumenti per il controllo ortografico e per la verifica della leggibilità sono altri esempi di strumenti di analisi statica.

3.1.1. Prodotti di Lavoro che possono essere esaminati dal Testing Statico

Quasi tutti i prodotti di lavoro possono essere esaminati utilizzando il testing statico. Esempi includono i documenti di specifica dei requisiti, il codice sorgente, i test plan, i test case, gli item del product backlog, i test charter, la documentazione di progetto, i contratti e i modelli.

Qualsiasi prodotto di lavoro che possa essere letto e compreso può essere oggetto di una review. Tuttavia, per l'analisi statica, i prodotti di lavoro devono avere una struttura rispetto alla quale possono essere verificati (es. modelli, codice o testo con una sintassi formale).

I prodotti di lavoro che non sono appropriati per il testing statico includono quelli che sono difficili da interpretare per gli esseri umani e che non dovrebbero essere analizzati dagli strumenti (es. il codice eseguibile di terze parti che non dovrebbe essere analizzato per motivi legali).

3.1.2. Valore del Testing Statico

Il testing statico può rilevare i difetti nelle prime fasi del SDLC, e questo soddisfa il principio del testing anticipato (si veda il paragrafo 1.3). Può anche identificare i difetti che non possono essere rilevati dal testing dinamico (es. codice irraggiungibile, pattern di progettazione non implementati come desiderato, difetti nei prodotti di lavoro non eseguibili).

Il testing statico fornisce la capacità di valutare la qualità e di creare fiducia nei prodotti di lavoro. Verificando i requisiti documentati, gli stakeholder possono anche assicurarsi che questi requisiti descrivano le loro esigenze effettive. Poiché il testing statico può essere eseguito nelle prime fasi del SDLC, è possibile creare una comprensione condivisa tra gli stakeholder coinvolti. Anche la comunicazione tra gli stakeholder coinvolti ne risulterà migliorata. Per questo motivo, si raccomanda di coinvolgere un'ampia varietà di stakeholder nel testing statico.

Anche se le review possono essere costose da implementare, i costi di progetto complessivi sono di solito molto inferiori rispetto a quando non vengono eseguite le review, perché deve essere dedicato meno tempo ed effort per la correzione dei difetti nelle fasi successive del progetto.

I difetti del codice possono essere rilevati utilizzando l'analisi statica in modo più efficiente rispetto al testing dinamico, e questo di solito si traduce in un minor numero di difetti del codice e in un minore effort di sviluppo complessivo.

3.1.3. Differenze tra Testing Statico e Testing Dinamico

Testing statico e testing dinamico sono complementari tra loro. Hanno obiettivi simili, come il supporto al rilevamento dei difetti nei prodotti di lavoro (si veda il paragrafo 1.1.1), ma esistono anche delle differenze, ad esempio:

- Il testing statico e dinamico (con l'analisi dei failure) possono entrambi portare alla rilevazione dei difetti, ma esistono alcuni tipi di difetti che possono essere rilevati solo dal testing statico o solo dal testing dinamico
- Il testing statico rileva direttamente i difetti nel prodotto di lavoro, mentre il testing dinamico genera failure da cui vengono identificati i difetti associati, attraverso un'analisi successiva
- Il testing statico può rilevare più facilmente i difetti presenti in percorsi del codice raramente esercitati o difficili da raggiungere con il testing dinamico
- Il testing statico può essere applicato a prodotti di lavoro non eseguibili, mentre il testing dinamico può essere applicato solo a prodotti di lavoro eseguibili
- Il testing statico può essere utilizzato per misurare le caratteristiche di qualità che sono indipendenti dall'esecuzione del codice (es. manutenibilità), mentre il testing dinamico può essere utilizzato per misurare le caratteristiche di qualità che sono dipendenti dall'esecuzione del codice (es. efficienza delle prestazioni).

I difetti tipici che sono più facili e/o più economici da rilevare attraverso il testing statico includono:

- Difetti nei requisiti (es. inconsistenze, ambiguità, contraddizioni, omissioni, inaccuratezze, duplicazioni)
- Difetti di progettazione (es. strutture di database inefficienti, scarsa modularità)
- Difetti di codifica (es. variabili con valori non definiti, variabili non dichiarate, codice non raggiungibile o duplicato, eccessiva complessità del codice)
- Deviazioni dagli standard (es. mancanza di aderenza alle naming convention degli standard di codifica)
- Specifiche di interfaccia non corrette (es. numero, tipo o ordine dei parametri non corrispondenti)
- Tipi specifici di vulnerabilità alla sicurezza (es. buffer overflow)
- Lacune o imprecisioni nella copertura della base di test (es. test mancanti per un criterio di accettazione)

3.2. Feedback e Processo di Review

3.2.1. Vantaggi di un Feedback Anticipato e Frequente degli Stakeholder

Un feedback anticipato e frequente permette di comunicare in anticipo i potenziali problemi di qualità. Se esiste poco coinvolgimento degli stakeholder durante il ciclo di vita dello sviluppo software, il prodotto che si sta sviluppando potrebbe non soddisfare la vision originale o attuale degli stakeholder. Il fallimento dovuto al rilascio di qualcosa che non corrisponde alle aspettative degli stakeholder, può causare costosi rework, scadenze non rispettate, rimbalzi di responsabilità e potrebbe persino portare al completo fallimento del progetto.

Un feedback frequente degli stakeholder durante l'intero SDLC può prevenire incomprensioni sui requisiti e assicurare che le modifiche ai requisiti siano comprese e implementate prima. Questo aiuta i membri del team di sviluppo a migliorare la comprensione rispetto a quello che stanno costruendo. Consente loro di concentrarsi sulle funzionalità che forniscono un maggior valore agli stakeholder e che hanno un impatto positivo sui rischi identificati.

3.2.2. Attività del Processo di Review

Lo standard ISO/IEC 20246 definisce un processo di review generico che fornisce un framework strutturato ma flessibile dal quale è possibile eseguire il tailoring (adattamento) di un processo di review specifico a una situazione particolare. Se la review richiesta è più formale, saranno necessari più task tra quelli descritti per le diverse attività.

La dimensione di molti prodotti di lavoro può essere tale da non poter essere coperta da un'unica review. Il processo di review può essere attivato un paio di volte per completare la review dell'intero prodotto di lavoro.

Le attività del processo di review generico sono:

- **Pianificazione.** Durante la fase di pianificazione, dovrà essere definito l'ambito della review, che comprende lo scopo, il prodotto di lavoro da sottoporre a review, le caratteristiche di qualità da valutare, le aree su cui focalizzarsi, i criteri di uscita, le informazioni di supporto come gli standard, l'effort e i tempi della review.
- **Inizio della review.** Durante l'inizio della review, l'obiettivo è assicurarsi che ogni persona coinvolta sia preparata per iniziare la review. Ciò significa assicurarsi che ogni partecipante abbia accesso al prodotto di lavoro sottoposto alla review, che comprenda il proprio ruolo e le proprie responsabilità, e che riceva tutto il necessario per eseguire la review.
- **Review individuale.** Ogni reviewer esegue una review individuale per valutare la qualità del prodotto di lavoro sottoposto alla review e per identificare anomalie, raccomandazioni e domande attraverso l'applicazione di una o più tecniche di review (es. review checklist-based, review scenario-based). Lo standard ISO/IEC 20246 approfondisce le diverse tecniche di review. I reviewer memorizzano tutte le anomalie, le raccomandazioni e le domande identificate.
- **Comunicazione e analisi.** Poiché le anomalie identificate durante una review non sono necessariamente difetti, tutte queste anomalie devono essere analizzate e discusse. Per ogni anomalia dovrebbe essere presa una decisione sullo stato, su chi debba prenderla in carico (proprietario) e sulle azioni necessarie. Questo avviene generalmente durante un review meeting,

in cui i partecipanti decidono anche quale sia il livello di qualità del prodotto di lavoro sottoposto alla review e quali azioni di follow-up siano richieste. Per completare le azioni può essere necessaria una review di follow-up.

- **Correzione e reporting.** Per ogni difetto, dovrebbe essere creato un defect report, in modo che le azioni correttive possano essere seguite. Una volta raggiunti i criteri di uscita, il prodotto di lavoro può essere accettato. Viene creato un report dei risultati della review.

3.2.3. Ruoli e Responsabilità nelle Review

Le review coinvolgono diversi stakeholder, che possono assumere diversi ruoli. I ruoli principali e le loro responsabilità sono:

- **Manager** - Decide cosa deve essere sottoposto a review e fornisce le risorse, come lo staff e il tempo per la review
- **Autore** - Crea e corregge il prodotto di lavoro sottoposto alla review
- **Moderatore** (chiamato anche facilitatore) - Garantisce l'esecuzione efficace dei review meeting, inclusa la mediazione, la gestione del tempo e un ambiente di review sicuro in cui tutti possano parlare liberamente
- **Scribe** (chiamato anche recorder) - Raccoglie le anomalie dai reviewer e memorizza le informazioni sulla review, come le decisioni e le nuove anomalie rilevate durante il review meeting
- **Reviewer** - Esegue le review. Un reviewer può essere una persona che lavora al progetto, un esperto in materia (subject matter expert) o qualsiasi altro stakeholder
- **Review leader** - Assume la responsabilità complessiva della review, decidendo ad esempio chi sarà coinvolto e organizzando quando e dove si svolgerà la review.

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

3.2.4. Tipi di Review

Esistono molti tipi di review, che possono variare da review informali a review formali. Il livello di formalità richiesto dipende da diversi fattori, come il ciclo di vita dello sviluppo software che deve essere applicato, la maturità del processo di sviluppo, la criticità e la complessità del prodotto di lavoro da sottoporre a review, i requisiti legali o normativi, e la necessità di un audit trail. Lo stesso prodotto di lavoro può essere sottoposto a diversi tipi di review, ad esempio applicando prima una review informale e successivamente una review più formale.

La selezione del tipo di review appropriata è fondamentale per raggiungere gli obiettivi di review richiesti (si veda il paragrafo 3.2.5). La selezione non si basa solo sugli obiettivi, ma anche su fattori come le esigenze di progetto, le risorse disponibili, il tipo di prodotto di lavoro e i relativi rischi, il dominio di business e la cultura aziendale.

Alcuni tipi di review comunemente utilizzati sono:

- **Review informale.** Le review informali non seguono un processo definito e non richiedono un risultato formale documentato. L'obiettivo principale è rilevare le anomalie.

- **Walkthrough.** Un walkthrough, che viene condotto dall'autore, può avere molti obiettivi, come valutare la qualità e creare fiducia nel prodotto di lavoro, educare i reviewer, ottenere il consenso, generare nuove idee, motivare e consentire agli autori di migliorare, rilevare anomalie. I reviewer possono eseguire una review individuale prima del walkthrough, ma non è obbligatoria.
- **Review tecnica.** Una review tecnica è eseguita da reviewer tecnicamente qualificati ed è guidata da un moderatore. Gli obiettivi di una review tecnica sono: ottenere il consenso e prendere decisioni riguardo un problema tecnico, rilevare anomalie, valutare la qualità e creare fiducia nel prodotto di lavoro, generare nuove idee, motivare, consentire agli autori di migliorare.
- **Ispezione.** Le ispezioni sono il tipo di review più formale e seguono il processo generico in modo completo (si veda il paragrafo 3.2.2). L'obiettivo principale è trovare il maggior numero di anomalie. Altri obiettivi sono: valutare la qualità, creare fiducia nel prodotto di lavoro, motivare e permettere agli autori di migliorare. Le metriche vengono raccolte e utilizzate per migliorare il ciclo di vita dello sviluppo software, incluso il processo di ispezione. Nelle ispezioni, l'autore non può agire come review leader o scribe della review.

3.2.5. Fattori di Successo per le Review

Esistono diversi fattori che determinano il successo delle review, che includono:

- Definire obiettivi chiari e criteri di uscita misurabili. La valutazione dei partecipanti non dovrebbe mai essere un obiettivo
- Scegliere il tipo di review appropriato per raggiungere gli obiettivi prefissati e per adattarsi al tipo di prodotto di lavoro, ai partecipanti alla review, alle esigenze e al contesto del progetto
- Condurre le review in piccole parti, in modo che i reviewer non perdano la concentrazione durante la review individuale e/o il review meeting (quando viene svolto)
- Fornire un feedback delle review agli stakeholder e agli autori, in modo che possano migliorare il prodotto e le loro attività (si veda il paragrafo 3.2.1)
- Fornire ai partecipanti un tempo adeguato per prepararsi alla review
- Avere il supporto del management per il processo di review
- Rendere le review parte della cultura dell'organizzazione, per promuovere l'apprendimento e il process improvement (miglioramento del processo)
- Fornire una formazione adeguata a tutti i partecipanti in modo che sappiano come svolgere il proprio ruolo
- Facilitare il review meeting

4. Analisi e Progettazione dei Test - 390 minuti

Parole chiave

acceptance test-driven development, analisi ai valori limite, approccio del test basato sulla collaborazione, copertura, copertura dei rami, copertura delle istruzioni, criteri di accettazione, elemento di copertura, error guessing, partizionamento di equivalenza, tecnica di test, tecnica di test basata sull'esperienza, tecnica di test black-box, tecnica di test white-box, testing checklist-based, testing della tabella delle decisioni, testing delle transizioni di stato, testing esplorativo

Obiettivi di Apprendimento per il Capitolo 4:

4.1 Panoramica sulle Tecniche di Test

FL-4.1.1 (K2) Distinguere le tecniche di test black-box, white-box e basate sull'esperienza

4.2 Tecniche di Test Black-box

FL-4.2.1 (K3) Utilizzare il partizionamento di equivalenza per derivare i test case

FL-4.2.2 (K3) Utilizzare l'analisi ai valori limite per derivare i test case

FL-4.2.3 (K3) Utilizzare il testing della tabella delle decisioni per derivare i test case

FL-4.2.4 (K3) Utilizzare il testing delle transizioni di stato per derivare i test case

4.3 Tecniche di Test White-box

FL-4.3.1 (K2) Spiegare il testing delle istruzioni

FL-4.3.2 (K2) Spiegare il testing dei rami

FL-4.3.3 (K2) Spiegare il valore del testing white-box

4.4 Tecniche di Test basate sull'Esperienza

FL-4.4.1 (K2) Spiegare l'error guessing

FL-4.4.2 (K2) Spiegare il testing esplorativo

FL-4.4.3 (K2) Spiegare il testing checklist-based

4.5. Approcci del Test basati sulla Collaborazione

FL-4.5.1 (K2) Spiegare come scrivere le user story in collaborazione con gli sviluppatori e i rappresentanti di business

FL-4.5.2 (K2) Classificare le differenti opzioni per la scrittura dei criteri di accettazione

FL-4.5.3 (K3) Utilizzare acceptance test-driven development (ATDD) per derivare i test case

4.1. Panoramica sulle Tecniche di Test

Le tecniche di test supportano il tester nell'analisi dei test (cosa testare) e nella progettazione dei test (come testare). Le tecniche di test aiutano a sviluppare un insieme relativamente piccolo, ma sufficiente, di test case in modo sistematico. Le tecniche di test aiutano inoltre il tester, durante l'analisi e la progettazione dei test, a definire le condizioni di test, a determinare gli elementi di copertura e a identificare i dati di test. Ulteriori informazioni sulle tecniche di test e sulle misure corrispondenti sono disponibili nello standard ISO/IEC/IEEE 29119-4 e in (Beizer 1990, Craig 2002, Copeland 2004, Koomen 2006, Jorgensen 2014, Ammann 2016, Forgács 2019).

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

Le **tecniche di test black-box** (note anche come tecniche basate sulle specifiche) si basano sull'analisi del comportamento specificato per l'oggetto di test senza fare riferimento alla sua struttura interna. Pertanto, i test case sono indipendenti da come è stato implementato il software. Di conseguenza, se l'implementazione viene modificata, ma il comportamento atteso rimane lo stesso, i test case sono ancora utili.

Le **tecniche di test white-box** (note anche come tecniche basate sulla struttura) si basano sull'analisi della struttura interna e dell'implementazione dell'oggetto di test. Poiché i test case dipendono da come è stato progettato il software, possono essere creati solo dopo la progettazione o l'implementazione dell'oggetto di test.

Le **tecniche di test basate sull'esperienza** utilizzano in modo efficace la conoscenza e l'esperienza dei tester per la progettazione e l'implementazione dei test case. L'efficacia di queste tecniche dipende in larga misura dalle competenze del tester. Le tecniche di test basate sull'esperienza possono rilevare difetti che potrebbero essere sfuggiti utilizzando le tecniche di test black-box e white-box. Pertanto, le tecniche di test basate sull'esperienza sono complementari alle tecniche di test black-box e white-box.

4.2. Tecniche di Test Black-Box

Le tecniche di test black-box comunemente utilizzate e trattate nei paragrafi successivi sono:

- Partizionamento di equivalenza
- Analisi ai valori limite
- Testing della tabella delle decisioni
- Testing delle transizioni di stato

4.2.1. Partizionamento di Equivalenza

Il partizionamento di equivalenza (EP, Equivalence Partitioning) divide i dati in partizioni (note come partizioni di equivalenza) basandosi sull'aspettativa che tutti gli elementi di una data partizione vengano elaborati nello stesso modo dall'oggetto di test. La teoria alla base di questa tecnica è che, se un test case che verifica un valore di una partizione di equivalenza rilevasse un difetto, questo difetto dovrebbe essere rilevato anche dai test case che verificano qualsiasi altro valore all'interno della stessa partizione.

Le partizioni di equivalenza possono essere identificate per qualsiasi elemento dei dati relativi all'oggetto di test, che includono gli input, gli output, i configuration item, i valori interni, i valori temporali e i parametri

di interfaccia. Le partizioni possono essere continue o discrete, ordinate o non ordinate, finite o infinite. Le partizioni non devono sovrapporsi e devono essere insiemi non vuoti.

Per oggetti di test semplici, il partizionamento di equivalenza può essere facile, ma in pratica, capire come l'oggetto di test tratterà i differenti valori è spesso complicato. Pertanto, il partizionamento dovrebbe essere fatto con attenzione.

Una partizione contenente valori validi è detta partizione valida. Una partizione che contiene valori non validi è detta partizione invalida. Le definizioni di valori validi e invalidi possono variare tra i team e le organizzazioni. Ad esempio, i valori validi possono essere interpretati come quelli che dovrebbero essere elaborati dall'oggetto di test o come quelli per i quali la specifica definisce la loro elaborazione. I valori invalidi possono essere interpretati come quelli che dovrebbero essere ignorati o rifiutati dall'oggetto di test o come quelli per i quali la specifica dell'oggetto di test non definisce alcuna elaborazione.

Nel partizionamento di equivalenza, gli elementi di copertura sono le partizioni di equivalenza. Per raggiungere il 100% di copertura con questa tecnica, i test case devono esercitare tutte le partizioni identificate (comprese quelle invalide) coprendo ogni partizione almeno una volta. La copertura, espressa in percentuale, è misurata come il numero di partizioni esercitate da almeno un test case, diviso per il numero totale di partizioni identificate.

Molti oggetti di test includono più insiemi di partizioni (es. oggetti di test con più di un parametro di input), e questo significa che un test case coprirà partizioni da differenti insiemi di partizioni. Il criterio di copertura più semplice nel caso di insiemi multipli di partizioni è chiamato copertura Each Choice (Ammann 2016). La copertura Each Choice richiede che i test case esercitino ogni partizione di ogni insieme di partizioni almeno una volta. La copertura Each Choice non tiene conto delle combinazioni di partizioni.

4.2.2. Analisi ai Valori Limite

L'analisi ai valori limite (BVA, Boundary Value Analysis) è una tecnica che si basa sull'esercitare i limiti delle partizioni di equivalenza. Pertanto, l'analisi ai valori limite può essere utilizzata solo per partizioni ordinate. I valori minimo e massimo di una partizione sono i suoi valori limite. Nell'analisi ai valori limite, se due elementi appartengono alla stessa partizione, anche tutti gli elementi compresi tra questi devono appartenere a quella partizione.

L'analisi ai valori limite si focalizza sui valori limite delle partizioni perché è più probabile che gli sviluppatori commettano errori con questi valori limite. I difetti tipici rilevati dall'analisi ai valori limite si localizzano dove i limiti implementati sono posizionati in modo errato, al di sopra o al di sotto della posizione prevista, o sono del tutto omessi.

Questo Syllabus copre due versioni dell'analisi ai valori limite: l'analisi ai valori limite a 2 valori (BVA a 2 valori) e l'analisi ai valori limite a 3 valori (BVA a 3 valori). Queste differiscono in termini di elementi di copertura per valore limite che devono essere esercitati per ottenere una copertura del 100%.

Nell'analisi ai **valori limite a 2 valori** (Craig 2002, Myers 2011), per ogni valore limite esistono due elementi di copertura: il valore limite e il suo valore adiacente più vicino appartenente alla partizione adiacente. Per raggiungere il 100% di copertura, i test case devono esercitare tutti gli elementi di copertura, cioè tutti i valori limite identificati. La copertura, espressa in percentuale, è misurata come il numero di valori limite esercitati, diviso per il numero totale di valori limite identificati.

Nell'analisi ai **valori limite a 3 valori** (Koomen 2006, O'Regan 2019), per ogni valore limite esistono tre elementi di copertura: il valore limite ed entrambi i suoi valori adiacenti. Pertanto, nell'analisi ai valori limite a 3 valori, alcuni elementi di copertura potrebbero non essere valori limite. Per ottenere il 100% di copertura,

i test case devono esercitare tutti gli elementi di copertura, cioè i valori limite identificati e i loro valori adiacenti. La copertura, espressa in percentuale, è misurata come il numero di valori limite e dei valori adiacenti esercitati, diviso per il numero totale di valori limite identificati e dei loro valori adiacenti.

L'analisi ai valori limite a 3 valori è più rigorosa dell'analisi ai valori limite a 2 valori, poiché può rilevare difetti trascurati dall'analisi ai valori limite a 2 valori. Ad esempio, se la decisione "IF (x ≤ 10) ..." è implementata in modo errato come "IF (x = 10) ...", nessun dato di test derivato dall'analisi ai valori limite a 2 valori (x = 10, x = 11) può rilevare il difetto. Tuttavia, x = 9, derivato dall'analisi ai valori limite a 3 valori, è probabile che venga rilevato.

4.2.3. Testing della Tabella delle Decisioni

Le tabelle delle decisioni sono utilizzate per il testing dell'implementazione dei requisiti di sistema che specificano come diverse combinazioni di condizioni producano risultati differenti. Le tabelle delle decisioni sono un modo efficace di registrare logiche complesse, come le regole di business.

Quando si creano le tabelle delle decisioni, vengono definite le condizioni e le azioni risultanti del sistema. Queste costituiscono le righe della tabella. Ogni colonna corrisponde a una regola decisionale che definisce una combinazione unica di condizioni, insieme alle azioni associate. Nelle tabelle delle decisioni limited-entry (a valori limitati) tutti i valori delle condizioni e delle azioni (eccetto quelli irrilevanti o non fattibili, per cui si veda di seguito) sono booleani, e quindi possono assumere solo i valori "vero" ("true") e "falso" ("false"). Nelle tabelle delle decisioni "extended-entry" (a valori estesi), alcuni o tutti i valori delle condizioni e delle azioni possono assumere valori multipli (es. valori discreti, intervalli numerici o partizioni di equivalenza).

La notazione per le condizioni è la seguente: "T" (true o vero) significa che la condizione è soddisfatta; "F" (false o falso) significa che la condizione non è soddisfatta; "-" significa che il valore della condizione è irrilevante per il risultato dell'azione; "N/A" ("not applicable") significa che la condizione non è fattibile per una determinata regola. La notazione per le azioni è la seguente: "X" significa che l'azione dovrebbe verificarsi; " " (blank o campo vuoto) significa che l'azione non dovrebbe verificarsi. Possono essere utilizzate anche altre notazioni.

Una tabella delle decisioni completa ha un numero di colonne sufficiente a coprire ogni combinazione di condizioni. La tabella può essere semplificata eliminando le colonne che contengono combinazioni di condizioni non fattibili. La tabella può anche essere minimizzata unendo in un'unica colonna tutte quelle colonne in cui alcune condizioni non influiscono sul risultato. Gli algoritmi di minimizzazione della tabella delle decisioni in una singola colonna non rientrano nell'ambito di questo Syllabus.

Nel testing della tabella delle decisioni, gli elementi di copertura sono le colonne contenenti le combinazioni fattibili di condizioni. Per ottenere una copertura del 100% con questa tecnica, i test case devono esercitare tutte queste colonne. La copertura, espressa in percentuale, è misurata come numero di colonne esercitate, diviso per il numero totale di colonne possibili.

Il punto di forza del testing della tabella delle decisioni è che fornisce un approccio sistematico per identificare tutte le combinazioni di condizioni, alcune delle quali potrebbero altrimenti essere trascurate. Inoltre, aiuta a trovare eventuali lacune o contraddizioni nei requisiti. Se le condizioni sono numerose, esercitare tutte le regole decisionali può richiedere molto tempo, poiché il numero di regole cresce esponenzialmente con il numero di condizioni. In questo caso, per ridurre il numero di regole che devono essere esercitate, può essere utilizzata una tabella delle decisioni minimizzata o un approccio basato sul rischio.

4.2.4. Testing delle Transizioni di Stato

Uno state transition diagram modella il comportamento di un sistema mostrando i suoi possibili stati e le transizioni di stato valide. Una transizione viene iniziata da un evento, che può essere ulteriormente qualificato da una condizione o “guard condition”. Si assume che le transizioni siano istantanee e che talvolta possano causare un’azione nel software. La sintassi comune per l'etichetta di una transizione è la seguente: "event [guard condition] / action". Le guard condition e le azioni possono essere omesse se non esistono o sono irrilevanti per il tester.

Una tabella degli stati è un modello equivalente allo state transition diagram. Le sue righe rappresentano gli stati e le sue colonne rappresentano gli eventi (insieme alle guard condition, se esistono). Le celle della tabella rappresentano le transizioni e contengono lo stato di destinazione e le azioni risultanti, se definite. A differenza dello state transition diagram, la tabella degli stati mostra esplicitamente le transizioni invalide, che sono rappresentate da celle vuote.

Un test case basato su uno state transition diagram o su una tabella degli stati è solitamente rappresentato come una sequenza di eventi, che si traduce in una sequenza di cambiamenti di stato (e di azioni, se necessario). Un test case può, e di solito lo farà, coprire diverse transizioni di stato.

Esistono molti criteri di copertura per il testing delle transizioni di stato. Questo Syllabus considera tre criteri di copertura.

Nella **copertura di tutti gli stati**, gli elementi di copertura sono gli stati. Per ottenere il 100% di copertura di tutti gli stati, i test case devono garantire che tutti gli stati siano visitati. La copertura, espressa in percentuale, è misurata come il numero di stati visitati diviso per il numero totale di stati.

Nella **copertura delle transizioni valide** (chiamata anche copertura 0-switch), gli elementi di copertura sono le singole transizioni valide. Per ottenere il 100% di copertura delle transizioni valide, i test case devono esercitare tutte le transizioni valide. La copertura, espressa in percentuale, è misurata come il numero di transizioni valide esercitate diviso per il numero totale di transizioni valide.

Nella **copertura di tutte le transizioni**, gli elementi di copertura sono tutte le transizioni mostrate in una tabella degli stati. Per ottenere il 100% di copertura di tutte le transizioni, i test case devono esercitare tutte le transizioni valide e tentare di eseguire le transizioni invalide. Eseguire il testing di una sola transizione invalida in un singolo test case aiuta a evitare il mascheramento dei difetti, cioè una situazione in cui un difetto impedisce il rilevamento di un altro. La copertura, espressa in percentuale, è misurata come il numero di transizioni valide e invalide esercitate o che si è tentato di coprire con i test case eseguiti, diviso per il numero totale di transizioni valide e invalide.

La copertura di tutti gli stati è più debole della copertura delle transizioni valide, perché in genere può essere ottenuta senza esercitare tutte le transizioni. La copertura delle transizioni valide è il criterio di copertura ampiamente più utilizzato. Raggiungere la copertura completa delle transizioni valide garantisce la copertura completa di tutti gli stati. Raggiungere la copertura completa di tutte le transizioni garantisce sia la copertura completa di tutti gli stati sia la copertura completa delle transizioni valide, e dovrebbe essere un requisito minimo per i software mission-critical e safety-critical.

4.3. Tecniche di Test White-Box

A causa della loro popolarità e semplicità, questo paragrafo si focalizza su due tecniche di test white-box relative al codice:

- Testing delle istruzioni

- Testing dei rami

Esistono tecniche più rigorose che vengono utilizzate in alcuni ambienti safety-critical, mission-critical o ad alta integrità per ottenere una copertura del codice più completa. Esistono anche tecniche di test white-box utilizzate nei livelli di test più elevati (es. testing delle API) o che utilizzano una copertura non correlata al codice (es. la copertura dei neuroni nel testing delle reti neurali). Queste tecniche non vengono discusse in questo Syllabus.

4.3.1. Testing delle Istruzioni e Copertura delle Istruzioni

Nel testing delle istruzioni, gli elementi di copertura sono le istruzioni eseguibili. L'obiettivo è progettare test case che esercitino le istruzioni nel codice fino a raggiungere un livello di copertura accettabile. La copertura, espressa in percentuale, è misurata come il numero di istruzioni esercitate dai test case diviso per il numero totale di istruzioni eseguibili nel codice.

Quando si raggiunge il 100% di copertura delle istruzioni, si garantisce che tutte le istruzioni eseguibili nel codice siano state esercitate almeno una volta. Questo significa che ogni istruzione contenente un difetto verrà eseguita e potrà causare un failure che dimostra la presenza del difetto, ma non significa che esercitare l'istruzione con un test case rileverà sempre il difetto. Ad esempio, potrebbero non essere rilevati difetti dipendenti dai dati (es. una divisione per zero che fallisce solo quando il denominatore è posto uguale a zero). Inoltre, la copertura del 100% delle istruzioni non garantisce che tutta la logica decisionale sia stata testata perché, ad esempio, potrebbe non essere possibile esercitare tutti i rami nel codice (si veda il capitolo 4.3.2).

4.3.2. Testing dei Rami e Copertura dei Rami

Un ramo (o “branch”) è un trasferimento di controllo tra due nodi nel grafo del flusso di controllo (control flow), che mostra le possibili sequenze in cui vengono eseguite le istruzioni del codice sorgente nell'oggetto di test. Ogni trasferimento di controllo può essere non condizionale (cioè codice lineare) o condizionale (cioè un risultato decisionale).

Nel testing dei rami, gli elementi di copertura sono i rami. L'obiettivo è progettare test case che esercitino i rami nel codice fino a raggiungere un livello di copertura accettabile. La copertura, espressa in percentuale, è misurata come il numero di rami esercitati dai test case diviso per il numero totale di rami.

Quando si raggiunge il 100% di copertura dei rami, tutti i rami del codice, non condizionali e condizionali, sono stati esercitati dai test case. I rami condizionali corrispondono tipicamente a un risultato vero o falso di una decisione "IF... THEN", a un risultato di un'istruzione SWITCH/CASE, o a una decisione di uscire o continuare in un ciclo (loop). Tuttavia, esercitare un ramo con un test case non rileverà sempre i difetti. Ad esempio, potrebbe non rilevare i difetti che richiedono l'esecuzione di un percorso specifico nel codice.

La copertura dei rami include la copertura delle istruzioni. Questo significa che qualsiasi insieme di test case che raggiunga il 100% di copertura dei rami raggiunge anche il 100% di copertura delle istruzioni (ma non viceversa).

4.3.3. Il Valore del Testing White-box

Un punto di forza fondamentale che tutte le tecniche white-box condividono è che durante il testing viene presa in considerazione l’implementazione completa del software, e questo facilita la rilevazione dei difetti anche quando la specifica del software è vaga, obsoleta o incompleta. Una debolezza è che se non

vengono implementati uno o più requisiti, il testing white-box potrebbe non rilevare i difetti risultanti dalla mancanza dei requisiti (Watson 1996).

Le tecniche white-box possono essere utilizzate nel testing statico (es. durante i dry run del codice). Sono adatte per eseguire la review del codice che non è ancora pronto per l'esecuzione (Hetzel 1988), oppure la review di pseudocodice e altre logiche di alto livello o top-down, che possono essere modellate con un grafo del flusso di controllo.

Eseguire solo il testing black-box non fornisce una misura dell'effettiva copertura del codice. Le misure della copertura white-box determinano una misura oggettiva della copertura e forniscono quelle informazioni necessarie per poter creare i test aggiuntivi che aumentino questa copertura e, di conseguenza, aumentino la fiducia nel codice.

4.4. Tecniche di Test basate sull'Esperienza

Le tecniche di test basate sull'esperienza comunemente utilizzate e trattate nei paragrafi successivi sono:

- Error guessing
- Testing esplorativo
- Testing checklist-based

4.4.1. Error Guessing

Error guessing è una tecnica utilizzata per anticipare il verificarsi di errori, difetti e failure, sulla base delle conoscenze del tester, che include:

- Come ha funzionato l'applicazione in passato
- I tipi di errori che gli sviluppatori tendono a commettere e i tipi di difetti che derivano da questi errori
- I tipi di failure che si sono verificati in altre applicazioni similari.

In generale, gli errori, i difetti e i failure possono essere relativi a: input (es. input corretto non accettato, parametri errati o mancanti), output (es. formato errato, risultato errato), logica (es. casi mancanti, operatore errato), calcolo (es. operando errato, calcolo errato), interfacce (es. mancata corrispondenza dei parametri, tipi incompatibili) o dati (es. inizializzazione errata, tipo errato).

I fault attack sono un approccio metodico all'implementazione dell'error guessing. Questa tecnica richiede che il tester crei o acquisisca un elenco di possibili errori, difetti e failure, e che progetti i test che identificheranno i difetti associati agli errori, rendano visibili i difetti, o causino i failure. Questi elenchi possono essere creati in base all'esperienza, ai dati sui difetti e sui failure, o alle conoscenze comuni sui motivi per cui il software fallisce.

Per ulteriori informazioni sull’error guessing e sui fault attack, si veda (Whittaker 2002, Whittaker 2003, Andrews 2006).

4.4.2. Testing Esplorativo

Nel testing esplorativo i test vengono progettati, eseguiti e valutati contemporaneamente, mentre il tester impara a conoscere l'oggetto di test. Il testing viene utilizzato per conoscere meglio l'oggetto di test, per esplorarlo in modo più approfondito con test focalizzati, e per creare i test per le aree non testate.

Il testing esplorativo viene talvolta condotto utilizzando il testing session-based per strutturare il testing. In un approccio session-based, il testing esplorativo viene svolto all'interno di un periodo time-box definito. Il tester utilizza un **Test Charter** contenente gli obiettivi del testing per guidare il testing. La sessione di test è solitamente seguita da un debriefing che prevede una discussione tra il tester e gli stakeholder interessati ai risultati della sessione di test. In questo approccio, gli obiettivi del testing possono essere trattati come condizioni di test di alto livello. Gli elementi di copertura sono identificati ed esercitati durante la sessione di test. Il tester può utilizzare dei session sheet di test per documentare i passi seguiti e le scoperte fatte.

Il testing esplorativo è utile quando le specifiche sono poche o inadeguate, oppure quando esistono pressioni significative sulle tempistiche del testing. Il testing esplorativo è anche utile per integrare altre tecniche di test più formali. Il testing esplorativo sarà più efficace se il tester è esperto, ha una conoscenza del dominio e possiede un alto grado di competenze essenziali, come competenze analitiche, curiosità e creatività (si veda il paragrafo 1.5.1).

Il testing esplorativo può incorporare l'uso di altre tecniche di test (es. il partizionamento di equivalenza). Ulteriori informazioni sul testing esplorativo sono disponibili in (Kaner 1999, Whittaker 2009, Hendrickson 2013).

4.4.3. Testing Checklist-Based

Nel testing checklist-based un tester progetta, implementa ed esegue i test per coprire le condizioni di test di una checklist. Le checklist possono essere costruite in base all'esperienza, alla conoscenza di ciò che è importante per l'utente, o alla comprensione del perché e del come il software fallisce. Le checklist non dovrebbero contenere elementi che possono essere controllati automaticamente, elementi che sono più adatti ad essere criteri di ingresso/uscita, o elementi troppo generici (Brykczynski 1999).

Gli elementi della checklist sono spesso formulati sotto forma di domanda. Dovrebbe essere possibile controllare ogni elemento separatamente e direttamente. Questi elementi possono riferirsi a requisiti, proprietà dell'interfaccia grafica, caratteristiche di qualità o altre forme di condizioni di test. Le checklist possono essere create per supportare vari tipi di test, compresi il testing funzionale e il testing non funzionale (es. le 10 euristiche per il testing di usabilità (Nielsen 1994)).

Alcuni elementi della checklist possono diventare gradualmente meno efficaci nel corso del tempo, perché gli sviluppatori impareranno a evitare di commettere gli stessi errori. Può anche essere necessario aggiungere nuovi elementi che corrispondano ai nuovi difetti di alta severità che sono stati rilevati. Pertanto, le checklist dovrebbero essere aggiornate regolarmente sulla base dell'analisi dei difetti. Bisogna comunque evitare che la checklist diventi troppo lunga (Gawande 2009).

In assenza di test case dettagliati, il testing checklist-based può fornire linee guida e un certo grado di consistenza per il testing. Se le checklist sono di alto livello, è probabile che si verifichi una certa variabilità nel testing effettivo, con una copertura potenzialmente maggiore ma una minore ripetibilità.

4.5. Approcci del Test basati sulla Collaborazione

Ognuna delle tecniche sopra descritte (si veda paragrafi 4.2, 4.3, 4.4) ha un obiettivo particolare rispetto al rilevamento dei difetti. Gli approcci basati sulla collaborazione, invece, si focalizzano anche sulla prevenzione dei difetti attraverso la collaborazione e la comunicazione.

4.5.1. Scrittura Collaborativa delle User Story

Una user story rappresenta una funzionalità che sarà significativa e importante per un utente o un cliente di un sistema o software. Le user story hanno tre aspetti critici (Jeffries 2000), chiamati insieme le "3 C":

- Card - Il mezzo che descrive una user story (es. una card, una voce in una card elettronica)
- Conversazione (Conversation) – La spiegazione di come verrà utilizzato il software (può essere scritta o verbale)
- Conferma (Confirmation) - I criteri di accettazione (si veda il paragrafo 4.5.2).

Il formato più comune per una user story è:

- In italiano: "Come [ruolo], voglio che [obiettivo sia realizzato], in modo da poter [valore di business risultante per il ruolo]", seguito dai criteri di accettazione
- In inglese: “As a [role], I want [goal to be accomplished], so that I can [resulting business value for the role]”, seguito dai criteri di accettazione.

La scrittura collaborativa delle user story può utilizzare tecniche come il brainstorming e le mappe mentali. La collaborazione consente al team di ottenere una visione condivisa di quello che dovrebbe essere rilasciato, tenendo conto di tre prospettive: business, sviluppo e testing.

Una buona user story dovrebbe rispettare il concetto di INVEST: Indipendente (independent), Negoziabile (Negotiable), Valutabile (Valuable), Stimabile (Estimable), Piccola (Small) e Testabile (Testable). Se uno stakeholder non conosce come testare una user story, questo può indicare che la user story non è abbastanza chiara, o che non riflette qualcosa di significativo per lo stakeholder, oppure che lo stakeholder ha semplicemente bisogno di un aiuto nel testing (Wake 2003).

4.5.2. Criteri di Accettazione

I criteri di accettazione per una user story sono le condizioni che un'implementazione della user story deve soddisfare per essere accettata dagli stakeholder. Da questo punto di vista, i criteri di accettazione possono essere visti come le condizioni di test che dovrebbero essere esercitate dai test. I criteri di accettazione sono di solito il risultato della Conversazione (Conversation) (si veda il paragrafo 4.5.1).

I criteri di accettazione sono utilizzati per:

- Definire l'ambito della user story
- Raggiungere il consenso tra gli stakeholder
- Descrivere scenari positivi e negativi
- Servire come base per il testing di accettazione delle user story (si veda il paragrafo 4.5.3)
- Consentire una pianificazione e una stima accurate

Esistono diversi modi per scrivere i criteri di accettazione di una user story. I due formati più comuni sono:

- Scenario-oriented (es. il formato Dato/Quando/Allora, o Given/When/Then, utilizzato nel BDD, si veda il paragrafo 2.1.3)
- Rule-oriented (es. un elenco puntato di verifiche o un formato tabellare della mappatura degli input-output)

La maggior parte dei criteri di accettazione può essere documentata in uno di questi due formati. Tuttavia, il team può utilizzare un altro formato custom, purché i criteri di accettazione siano ben definiti e non ambigui.

4.5.3. Acceptance Test-Driven Development (ATDD)

Acceptance Test-Driven Development (ATDD) è un approccio test-first (si veda il paragrafo 2.1.3). I test case vengono creati prima dell'implementazione della user story. I test case sono creati da membri del team con prospettive differenti, ad esempio clienti, sviluppatori e tester (Adzic 2009). I test case possono essere eseguiti manualmente o in modo automatizzato.

Il primo passo è un workshop di specifica in cui la user story e i relativi criteri di accettazione (se non ancora definiti) vengono analizzati, discussi e scritti dai membri del team. Durante questo processo vengono risolte le incompletezze, le ambiguità o i difetti nella user story. Il passo successivo è la creazione dei test case. Questo può essere eseguito da tutto il team o dal singolo tester. I test case sono basati sui criteri di accettazione e possono essere visti come esempi di come funziona il software. Questo aiuterà il team a implementare correttamente la user story.

Poiché esempi e test sono la stessa cosa, questi termini sono spesso usati in modo intercambiabile. Durante la progettazione dei test possono essere applicate le tecniche di test descritte nei paragrafi 4.2, 4.3 e 4.4.

In genere, i primi test case sono positivi, confermano quindi il comportamento corretto senza eccezioni o condizioni di errore, e comprendono la sequenza di attività eseguite se tutto va come previsto. Dopo aver completato i test case positivi, il team dovrebbe eseguire il testing negativo. Infine, il team dovrebbe coprire le caratteristiche di qualità non funzionali (es. efficienza delle prestazioni, usabilità). I test case dovrebbero essere espressi in modo comprensibile per gli stakeholder. In genere, i test case contengono frasi nel linguaggio naturale che includono le precondizioni necessarie (se presenti), gli input e le postcondizioni.

I test case devono coprire tutte le caratteristiche della user story e non dovrebbero andare oltre la user story. Tuttavia, i criteri di accettazione possono dettagliare alcuni degli argomenti descritti nella user story. Inoltre, nessun test case dovrebbe descrivere le stesse caratteristiche della user story.

Quando vengono acquisiti in un formato supportato da un framework di test automation, gli sviluppatori possono automatizzare i test case, scrivendo il codice di supporto durante l'implementazione della funzionalità descritta da una user story. I test di accettazione diventano quindi requisiti eseguibili.

5. Gestione delle Attività di Test - 335 minuti

Parole chiave

analisi del rischio, approccio del test, controllo dei test, controllo del rischio, criteri di ingresso, criteri di uscita, defect management, defect report, identificazione del rischio, livello di rischio, mitigazione del rischio, monitoraggio dei test, monitoraggio del rischio, pianificazione dei test, piramide di test, quadranti del testing, rischio, rischio di prodotto, rischio di progetto, risk management, testing basato sul rischio, test completion report, test plan, test progress report, valutazione del rischio

Obiettivi di Apprendimento per il Capitolo 5:

5.1 Pianificazione dei Test

FL-5.1.1 (K2) Fornire esempi sullo scopo e sul contenuto di un test plan

FL-5.1.2 (K1) Riconoscere come un tester aggiunge valore alla pianificazione dell'iterazione (iteration planning) alla pianificazione della release (release planning)

FL-5.1.3 (K2) Confrontare e mettere in contrapposizione i criteri di ingresso e i criteri di uscita

FL-5.1.4 (K3) Utilizzare tecniche di stima per calcolare l'effort del test richiesto

FL-5.1.5 (K3) Applicare la prioritizzazione dei test case

FL-5.1.6 (K1) Richiamare i concetti della piramide di test

FL-5.1.7 (K2) Riassumere i quadranti del testing e le loro relazioni con i livelli di test e i tipi di test

5.2 Risk Management

FL-5.2.1 (K1) Identificare il livello di rischio utilizzando la probabilità del rischio e l'impatto del rischio

FL-5.2.2 (K2) Distinguere tra rischi di progetto e rischi di prodotto

FL-5.2.3 (K2) Spiegare come l'analisi dei rischi di prodotto possa influenzare la completezza e l'ambito del testing

FL-5.2.4 (K2) Spiegare quali misure possono essere adottate in risposta ai rischi di prodotto analizzati

5.3 Monitoraggio dei Test, Controllo dei Test e Completamento dei Test

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

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

FL-5.3.3 (K2) Fornire esempi su come comunicare lo stato di avanzamento del testing

5.4 Configuration Management

FL-5.4.1 (K2) Riassumere come il configuration management supporta il testing

5.5 Defect Management

FL-5.5.1 (K3) Preparare un defect report

5.1. Pianificazione dei Test

5.1.1. Scopo e Contenuto di un Test Plan

Un test plan descrive gli obiettivi, le risorse e i processi di un progetto di test. Un test plan:

- Documenta i mezzi e la schedulazione per raggiungere gli obiettivi del testing
- Contribuisce a garantire che le attività di test eseguite soddisferanno i criteri stabiliti
- Serve come mezzo di comunicazione con i membri del team e gli altri stakeholder
- Dimostra che il testing aderirà alla politica di test e alla strategia di test esistenti (o spiega perché il testing devia da questi).

La pianificazione dei test guida il pensiero dei tester e li costringe a confrontarsi con le sfide future relative ai rischi, alle schedulazioni, alle persone, agli strumenti, ai costi, agli effort, ecc. Il processo di preparazione di un test plan è un modo utile per riflettere sull'effort necessario per raggiungere gli obiettivi del progetto di test.

Il contenuto tipico di un test plan include:

- Contesto del testing (es. ambito, obiettivi del testing, vincoli, base di test)
- Assunzioni e vincoli del progetto di test
- Stakeholder (es. ruoli, responsabilità, rilevanza per il testing, esigenze di assunzione e formazione)
- Comunicazione (es. forme e frequenza di comunicazione, template di documentazione)
- Risk Register (es. rischi di prodotto, rischi di progetto)
- Approccio del test (es. livelli di test, tipi di test, tecniche di test, prodotti del test, criteri di ingresso e criteri di uscita, indipendenza del testing, metriche da raccogliere, requisiti dei dati di test, requisiti dell'ambiente di test, deviazioni dalla politica di test e dalla strategia di test dell'organizzazione)
- Budget e schedulazione

Maggiori dettagli sul test plan e sul suo contenuto possono essere trovati nello standard ISO/IEC/IEEE 29119-3.

5.1.2. Contributo del Tester alla Pianificazione dell'Iterazione e alla Pianificazione della Release

Nei cicli di vita dello sviluppo software iterativi, in genere si verificano due tipi di pianificazione: la pianificazione della release (release planning) e la pianificazione dell'iterazione (iteration planning).

La pianificazione della release guarda alla release di un prodotto, definisce e ridefinisce il product backlog e può comportare il raffinamento delle user story più grandi in un insieme di user story più piccole. Serve anche come base per l'approccio del test e per il test plan in tutte le iterazioni. I tester coinvolti nella pianificazione della release partecipano alla scrittura delle user story e di criteri di accettazione testabili (si veda il paragrafo 4.5), partecipano all'analisi dei rischi di progetto e di qualità (si veda il paragrafo 5.2), stimano l'effort del test associato alle user story (si veda il paragrafo 5.1.4), determinano l'approccio del test e pianificano il testing per la release.

La pianificazione dell'iterazione guarda alla fine di una singola iterazione e si occupa dell'iteration backlog. I tester coinvolti nella pianificazione dell'iterazione partecipano all'analisi dettagliata dei rischi delle user story, determinano la testabilità delle user story, suddividono le user story in task (attività), in particolare i task del testing, stimano l'effort per tutti i task del testing, identificano e raffinano gli aspetti funzionali e non funzionali dell'oggetto di test.

5.1.3. Criteri di Ingresso e Criteri di Uscita

I criteri di ingresso definiscono le precondizioni per iniziare una determinata attività. Se i criteri di ingresso non sono soddisfatti, è probabile che l'attività si riveli essere più difficile, dispendiosa in termini di tempi e costi, e rischiosa. I criteri di uscita definiscono che cosa deve essere raggiunto per dichiarare un'attività completata. I criteri di ingresso e i criteri di uscita dovrebbero essere definiti per ogni livello di test, e variano in base agli obiettivi del testing.

Criteri di ingresso tipici includono: disponibilità delle risorse (es. persone, strumenti, ambienti, dati di test, budget, tempo), disponibilità del testware (es. base di test, requisiti testabili, user story, test case) e livello di qualità iniziale di un oggetto di test (es. tutti gli smoke test sono stati superati).

Criteri di uscita tipici includono: misure di completezza (es. livello di copertura raggiunto, numero di difetti non risolti, densità dei difetti, numero di test case falliti) e criteri di completamento (es. i test pianificati sono stati eseguiti, il testing statico è stato eseguito, tutti i difetti trovati sono stati segnalati, tutti i regression test sono stati automatizzati).

Anche l'esaurimento del tempo disponibile o del budget possono essere considerati criteri di uscita validi. Anche senza che siano soddisfatti altri criteri di uscita, può essere accettabile terminare il testing in queste circostanze, se gli stakeholder hanno eseguito la review e accettato il rischio di andare in produzione senza ulteriore testing.

Nello sviluppo software Agile, i criteri di uscita sono spesso chiamati Definition of Done, e definiscono le metriche oggettive del team per un elemento rilasciabile. I criteri di ingresso che una user story deve soddisfare per iniziare le attività di sviluppo e/o di test sono chiamati Definition of Ready.

5.1.4. Tecniche di Stima

La stima dell'effort del test consiste nel prevedere la quantità di lavoro relativo al testing necessario per raggiungere gli obiettivi di un progetto di test. È importante chiarire agli stakeholder che la stima si basa su una serie di assunzioni ed è sempre soggetta a errori di stima. La stima per piccoli task è generalmente più accurata di quella per grandi task. Pertanto, quando si stima un task di grandi dimensioni, può essere decomposto in un insieme di task più piccoli, che a loro volta possono essere stimati.

In questo Syllabus vengono descritte le seguenti quattro tecniche di stima.

Stima basata su valori statistici. In questa tecnica metrics-based (basata su metriche) i dati vengono raccolti da progetti precedenti all'interno dell'organizzazione, e questo permette di derivare valori statistici "standard" per progetti similari. I valori statistici dei progetti dell'organizzazione (es. ricavati dai dati storici) sono generalmente la fonte migliore da utilizzare nel processo di stima. Queste statistiche standard possono poi essere utilizzate per stimare l'effort del test per il nuovo progetto. Ad esempio, se nel progetto precedente il rapporto dell'effort dello sviluppo rispetto all'effort del test era di 3:2, e nel progetto attuale si prevede che l'effort dello sviluppo sia 600 giorni-persona, allora l'effort del test può essere stimato in 400 giorni-persona.

Estrapolazione. In questa tecnica metrics-based le misure vengono effettuate nelle prime fasi del progetto attuale per poter raccogliere i dati. Avendo un numero sufficiente di osservazioni, l'effort richiesto per il lavoro rimanente può essere approssimato estrapolando questi dati (di solito applicando un modello matematico). Questo metodo è molto adatto ai cicli di vita iterativi. Ad esempio, il team può estrapolare l'effort del test nella prossima iterazione come valore medio dell'effort relativo alle ultime tre iterazioni.

Wideband Delphi. In questa tecnica expert-based (basata su esperti) iterativa, gli esperti effettuano stime basate sull'esperienza. Ogni esperto stima l'effort in autonomia. I risultati vengono raccolti e se esistono deviazioni che non rientrano nei limiti concordati, gli esperti discutono le loro stime attuali. A ciascun esperto viene quindi chiesto di effettuare una nuova stima sulla base di questo feedback, sempre in autonomia. Questo processo viene ripetuto finché non si raggiunge un consenso. Planning Poker è una variante di Wideband Delphi, comunemente utilizzata nello sviluppo software Agile. Nel Planning Poker le stime vengono solitamente effettuate utilizzando carte con numeri che rappresentano la dimensione dell'effort.

Three-point estimation (stima dei tre punti). In questa tecnica expert-based, gli esperti effettuano tre stime: la stima più ottimistica (a), la stima più probabile (m) e la stima più pessimistica (b). La stima finale (E) è la loro media aritmetica pesata. Nella versione più popolare di questa tecnica, la stima viene calcolata come E = (a + 4*m + b) / 6. Il vantaggio di questa tecnica è che permette agli esperti di calcolare l'errore di misura: SD = (b - a) / 6. Ad esempio, se le stime (in ore-persona) sono: a=6, m=9 e b=18, la stima finale è di 10 ± 2 ore-persona (cioè, tra 8 e 12 ore-persona), perché

E = (6 + 4*9 + 18) / 6 = 10

SD = (18 - 6) / 6 = 2.

Per queste e molte altre tecniche di stima del test, si veda (Kan 2003, Koomen 2006, Westfall 2009).

5.1.5. Prioritizzazione dei Test Case

Una volta che i test case e le procedure di test vengono specificati e organizzati in test suite, queste test suite possono essere organizzate in una schedulazione di esecuzione dei test che definisce l'ordine in cui devono essere eseguite. Quando si prioritizzano i test case, possono essere presi in considerazione diversi fattori. Le strategie di prioritizzazione dei test case più comunemente utilizzate sono le seguenti:

- **Prioritizzazione basata sul rischio**, dove l'ordine di esecuzione dei test si basa sui risultati dell'analisi del rischio (si veda il paragrafo 5.2.3). I test case che coprono i rischi più importanti vengono eseguiti per primi.
- **Prioritizzazione basata sulla copertura**, dove l'ordine di esecuzione dei test si basa sulla copertura (es. la copertura delle istruzioni). I test case che raggiungono la copertura più elevata vengono eseguiti per primi. In un'altra variante, chiamata prioritizzazione della copertura aggiuntiva, il test case che raggiunge la copertura più alta viene eseguito per primo; ogni test case successivo è quello che raggiunge la copertura aggiuntiva più elevata.
- **Prioritizzazione basata sui requisiti**, dove l'ordine di esecuzione dei test si basa sulle priorità dei requisiti tracciati rispetto ai test case corrispondenti. Le priorità dei requisiti vengono definite dagli stakeholder. I test case relativi ai requisiti più importanti vengono eseguiti per primi.

Idealmente, i test case dovrebbero essere ordinati per l'esecuzione in base al loro livello di priorità utilizzando, ad esempio, una delle strategie di prioritizzazione sopra menzionate. Tuttavia, questa pratica può non funzionare se i test case o le funzionalità da testare hanno delle dipendenze. Se un test case con priorità più alta dipende da un test case con priorità più bassa, il test case con priorità più bassa deve essere eseguito prima.

L'ordine di esecuzione dei test deve tenere conto anche della disponibilità delle risorse. Ad esempio, gli strumenti di test richiesti, gli ambienti di test o le persone che possono essere disponibili solo per una specifica finestra temporale.

5.1.6. La Piramide di Test

La piramide di test è un modello che mostra che test differenti possono avere granularità differenti. Il modello della piramide di test supporta il team nella test automation e nell'allocazione dell'effort del test, mostrando che obiettivi differenti sono supportati da livelli differenti di test automation. I livelli della piramide rappresentano gruppi di test. Più alto è il livello, minori sono la granularità del test, l'isolamento del test e i tempi di esecuzione dei test. I test del livello inferiore sono piccoli, isolati, veloci e verificano una piccola parte della funzionalità; quindi, di solito servono molti test per ottenere una copertura ragionevole. Il livello superiore rappresenta test complessi, di alto livello, end-to-end. Questi test di alto livello sono in genere più lenti dei test dei livelli inferiori e verificano di solito una gran parte della funzionalità; quindi, di solito bastano pochi test per ottenere una copertura ragionevole. Il numero e la denominazione dei livelli possono variare. Ad esempio, il modello originale della piramide di test (Cohn 2009) definisce tre livelli: "unit test", "service test" e "UI test". Un altro modello molto diffuso definisce unit test (test di componente), test di integrazione (dei componenti) e test end-to-end. Si possono utilizzare anche altri livelli di test (si veda il paragrafo 2.2.1).

5.1.7. Quadranti del Testing

I quadranti del testing, definiti da Brian Marick (Marick 2003, Crispin 2008), raggruppano i livelli di test con i tipi di test, le attività, le tecniche di test e i prodotti di lavoro appropriati nello sviluppo software Agile. Il modello supporta il test management nel visualizzarli, per garantire che tutti i tipi di test e i livelli di test appropriati siano inclusi nel ciclo di vita dello sviluppo software, e nel capire che alcuni tipi di test sono più rilevanti per determinati livelli di test rispetto ad altri. Questo modello fornisce anche un modo per differenziare e descrivere i tipi di test a tutti gli stakeholder, compresi gli sviluppatori, i tester e i rappresentanti di business.

In questo modello, i test possono essere orientati al business oppure orientati alla tecnologia. I test possono anche supportare il team (cioè guidare lo sviluppo) o criticare il prodotto (cioè misurare il suo comportamento rispetto alle aspettative). La combinazione di questi due punti di vista determina i quattro quadranti:

- **Quadrante Q1** (orientato alla tecnologia e a supporto del team). Questo quadrante contiene test di componente e test di integrazione dei componenti. Questi test dovrebbero essere automatizzati e inclusi nel processo di Continuous Integration.
- **Quadrante Q2** (orientato al business e a supporto del team). Questo quadrante contiene test funzionali, esempi, test di user story, prototipi di user experience (UX), testing delle API e simulazioni. Questi test verificano i criteri di accettazione e possono essere manuali o automatizzati.
- **Quadrante Q3** (orientato al business e a critica del prodotto). Questo quadrante contiene test esplorativi, test di usabilità, e user acceptance test. Questi test sono orientati all'utente e sono spesso manuali.
- **Quadrante Q4** (orientato alla tecnologia e a critica del prodotto). Questo quadrante contiene smoke test e test non funzionali (eccetto i test di usabilità). Questi test sono spesso automatizzati.

5.2. Risk Management

Le organizzazioni devono affrontare molti fattori interni ed esterni che rendono incerto se e quando raggiungeranno i loro obiettivi (ISO 31000). Il risk management permette alle organizzazioni di aumentare la probabilità di raggiungere gli obiettivi, migliorare la qualità dei loro prodotti e aumentare la confidenza e la fiducia degli stakeholder.

Le principali attività di risk management sono:

- **Analisi del rischio** (che consiste nell'identificazione del rischio e nella valutazione del rischio; si veda il paragrafo 5.2.3)
- **Controllo del rischio** (che consiste nella mitigazione del rischio e nel monitoraggio del rischio; si veda il paragrafo 5.2.4)

L'approccio del test, in cui vengono selezionate, prioritizzate e gestite le attività di test in base all'analisi del rischio e al controllo del rischio, è chiamato testing basato sul rischio.

5.2.1. Definizione di Rischio e Attributi del Rischio

Il rischio è un evento potenziale, un pericolo (hazard), una minaccia o una situazione il cui verificarsi provoca un effetto negativo. Un rischio può essere caratterizzato da due fattori:

- **Probabilità del rischio** - La probabilità di accadimento del rischio (maggiore di zero e minore di uno)
- **Impatto del rischio** (danno) - Le conseguenze di questo accadimento

Questi due fattori esprimono il livello di rischio, che è una misura del rischio. Più alto è il livello di rischio, più importante è la sua gestione.

5.2.2. Rischi di Progetto e Rischi di Prodotto

Nel testing del software vengono considerati generalmente due tipi di rischio: rischi di progetto e i rischi di prodotto.

I rischi di progetto sono relativi alla gestione e al controllo del progetto. I rischi di progetto includono:

- Problemi organizzativi (es. ritardi nel rilascio dei prodotti di lavoro, stime non accurate, tagli dei costi)
- Problemi legati alle persone (es. competenze insufficienti, conflitti, problemi di comunicazione, carenza di personale)
- Problemi tecnici (es. scope creep, scarso supporto degli strumenti)
- Problemi con i fornitori (es. mancato rilascio da terze parti, fallimento dell'azienda fornitrice)

I rischi di progetto, quando si verificano, possono avere un impatto sulla schedulazione, sul budget o sull'ambito del progetto, che influisce sulla capacità del progetto di raggiungere gli obiettivi.

I rischi di prodotto sono relativi alle caratteristiche di qualità del prodotto (es. descritte nel modello di qualità ISO 25010). Esempi di rischi di prodotto sono: funzionalità mancanti o errate, calcoli errati, errori di runtime, architettura insufficiente, algoritmi inefficienti, tempi di risposta non adeguati, scarsa user

experience (UX), vulnerabilità della sicurezza. I rischi di prodotto, quando si verificano, possono portare a diverse conseguenze negative, che includono:

- Insoddisfazione degli utenti
- Perdita di fatturato, fiducia e reputazione
- Danni a terze parti
- Elevati costi di manutenzione, sovraccarico dell'helpdesk
- Sanzioni penali
- In casi estremi, danni fisici, lesioni o addirittura morte

5.2.3. Analisi dei Rischi di Prodotto

Dal punto di vista del testing, l'obiettivo dell'analisi dei rischi di prodotto è quello di fornire una consapevolezza del rischio del prodotto, per poter focalizzare l'effort del testing in modo da minimizzare il livello residuo del rischio di prodotto. Idealmente, l'analisi dei rischi di prodotto inizia nelle prime fasi del ciclo di vita dello sviluppo software.

L'analisi dei rischi di prodotto consiste nell'identificazione del rischio e nella valutazione del rischio. L'identificazione del rischio consiste nel generare un elenco completo dei rischi. Gli stakeholder possono identificare i rischi utilizzando diverse tecniche e strumenti, per esempio brainstorming, workshop, interviste o diagrammi causa-effetto. La valutazione del rischio comporta: la categorizzazione dei rischi identificati, la valutazione della probabilità, dell'impatto e del livello di rischio, la prioritizzazione dei rischi e la proposta di azioni per gestirli. La categorizzazione aiuta ad assegnare le azioni di mitigazione, perché di solito i rischi che rientrano nella stessa categoria possono essere mitigati utilizzando un approccio similare.

La valutazione del rischio può utilizzare un approccio quantitativo o qualitativo, o un mix di questi. Nell'approccio quantitativo il livello di rischio è calcolato come moltiplicazione della probabilità del rischio e dell'impatto del rischio. Nell'approccio qualitativo il livello di rischio può essere determinato utilizzando una matrice del rischio.

L'analisi dei rischi di prodotto può influenzare l'accuratezza e l'ambito del testing. I suoi risultati vengono utilizzati per:

- Determinare l'ambito del testing da eseguire
- Determinare i livelli di test particolari e proporre i tipi di test da eseguire
- Determinare le tecniche di test da applicare e la copertura da raggiungere
- Stimare l'effort del test richiesto per ogni attività
- Prioritizzare il testing nel tentativo di rilevare i difetti critici il più presto possibile
- Determinare se in aggiunta al testing potrebbero essere utilizzate attività aggiuntive per ridurre il rischio

5.2.4. Controllo dei Rischi di Prodotto

Il controllo dei rischi di prodotto comprende tutte le misure che vengono prese in risposta ai rischi di prodotto identificati e valutati. Il controllo dei rischi di prodotto consiste nella mitigazione del rischio e nel monitoraggio del rischio. La mitigazione del rischio comporta l'implementazione delle azioni proposte durante la valutazione del rischio, per ridurre il livello di rischio. L'obiettivo del monitoraggio del rischio è garantire che le azioni di mitigazione siano efficaci, ottenere ulteriori informazioni per migliorare la valutazione del rischio, e identificare i rischi emergenti.

Una volta analizzato il rischio, sono possibili diverse opzioni di risposta al rischio, per esempio la mitigazione del rischio attraverso il testing, l'accettazione del rischio, il trasferimento del rischio o il contingency plan (Veenendaal 2012). Le azioni che possono essere intraprese per mitigare i rischi di prodotto attraverso il testing sono le seguenti:

- Selezionare i tester con il giusto livello di esperienza e competenza, adeguati a un determinato tipo di rischio
- Applicare un livello appropriato di indipendenza del testing
- Condurre review ed eseguire l’analisi statica
- Applicare le tecniche di test e i livelli di copertura appropriati
- Applicare i tipi di test più appropriati per le caratteristiche di qualità desiderate
- Eseguire il testing dinamico, che include il regression testing

5.3. Monitoraggio dei Test, Controllo dei Test e Completamento dei Test

Il monitoraggio dei test si occupa di raccogliere informazioni sul testing. Queste informazioni vengono utilizzate per valutare l'avanzamento del test e per misurare se i criteri di uscita del test o le attività di test associate ai criteri di uscita sono stati soddisfatti, come il raggiungimento degli obiettivi di copertura dei rischi di prodotto, dei requisiti o dei criteri di accettazione.

Il controllo dei test utilizza le informazioni ottenute dal monitoraggio dei test per fornire, sotto forma di direttive di controllo, una guida e le azioni correttive necessarie per ottenere il testing più efficace ed efficiente. Esempi di direttive di controllo includono:

- Riprioritizzare i test quando un rischio identificato diventa un problema
- Rivalutare se un elemento di test soddisfa i criteri di ingresso o i criteri di uscita a causa di un rework
- Adattare la schedulazione dei test per indirizzare un ritardo nel rilascio dell'ambiente di test
- Aggiungere nuove risorse quando e dove necessario

Il completamento dei test raccoglie i dati delle attività di test completate per consolidare l'esperienza, il testware e qualsiasi altra informazione rilevante. Le attività di completamento dei test vengono eseguite al raggiungimento delle milestone di progetto, come il completamento di un livello di test, la conclusione di un'iterazione Agile, il completamento (o la cancellazione) di un progetto di test, il rilascio di un sistema software o il completamento di una release di manutenzione.

5.3.1. Metriche Utilizzate nel Testing

Le metriche di test vengono raccolte per valutare l'avanzamento rispetto alla schedulazione e al budget pianificati, la qualità attuale dell'oggetto di test, e l'efficacia delle attività di test rispetto allo scopo di un'iterazione oppure agli obiettivi. Il monitoraggio dei test colleziona una varietà di metriche per supportare il controllo dei test e il completamento dei test.

Le metriche di test comuni includono:

- Metriche di avanzamento del progetto (es. completamento dei task, utilizzo delle risorse, effort del test)
- Metriche di avanzamento del test (es. avanzamento dell'implementazione dei test case, avanzamento della preparazione dell'ambiente di test, numero di test case eseguiti/non eseguiti, superati/falliti, tempo di esecuzione dei test)
- Metriche di qualità del prodotto (es. disponibilità, tempi di risposta, mean time to failure)
- Metriche dei difetti (es. numero e priorità dei difetti trovati/risolti, densità dei difetti, percentuale di rilevamento dei difetti)
- Metriche di rischio (es. livello di rischio residuo)
- Metriche di copertura (es. copertura dei requisiti, copertura del codice)
- Metriche di costo (es. costo del testing, costo della qualità dell'organizzazione).

5.3.2. Scopo, Contenuto e Destinatari dei Test Report

La reportistica dei test riassume e comunica le informazioni dei test durante e dopo il testing. I test progress report supportano il controllo continuo del testing e devono fornire informazioni sufficienti per apportare modifiche alla schedulazione dei test, alle risorse o al test plan, quando tali modifiche sono necessarie a causa di deviazioni dal piano o di circostanze mutate. I test completion report riassumono una fase specifica del testing (es. livello di test, ciclo di test, iterazione) e possono fornire informazioni per il testing successivo.

Durante il monitoraggio e il controllo dei test, il team di test genera test progress report per gli stakeholder, per tenerli informati. I test progress report sono solitamente generati su base regolare (es. giornalmente, settimanalmente, ecc.) e includono:

- Periodo di test
- Stato di avanzamento del test (es. in anticipo o in ritardo rispetto alla schedulazione), incluse eventuali deviazioni degne di nota
- Impedimenti per il testing e relativi workaround
- Metriche di test (si veda il paragrafo 5.3.1 per gli esempi)
- Rischi nuovi e modificati durante il periodo del testing
- Testing pianificato per il prossimo periodo

Un **test completion report** viene preparato durante il completamento dei test, quando un progetto, un livello di test o un tipo di test è completo e quando, idealmente, i criteri di uscita sono stati soddisfatti. Questo report utilizza i test progress report e altri dati. I tipici test completion report includono:

- Riassunto del test
- Valutazione del testing e della qualità del prodotto in base al test plan originale (cioè, obiettivi del testing e criteri di uscita)
- Deviazioni dal test plan (es. differenze rispetto alla schedulazione, alla durata e all'effort pianificati)
- Testing degli impedimenti e workaround
- Metriche di test basate sui test progress report
- Rischi non mitigati, difetti non risolti
- Lessons learned rilevanti per il testing

I diversi destinatari richiedono informazioni differenti nei report, e influenzano il grado di formalità e la frequenza della reportistica. La reportistica dell'avanzamento del test agli altri membri dello stesso team è spesso frequente e informale, mentre la reportistica sul testing per un progetto completato segue un template prestabilito e avviene una volta sola.

Lo standard ISO/IEC/IEEE 29119-3 include template ed esempi per i test progress report (chiamati test status report) e test completion report.

5.3.3. Comunicare lo Stato del Testing

Il mezzo migliore per comunicare lo stato del testing varia, in base alle esigenze di test management, alle strategie di test dell'organizzazione, agli standard normativi o, in caso di team auto-organizzati (si veda il paragrafo 1.5.2), al team stesso. Le opzioni includono:

- Comunicazione verbale con i membri del team e gli altri stakeholder
- Dashboard (es. dashboard CI/CD, taskboard e burn-down chart)
- Canali di comunicazione elettronica (es. e-mail, chat)
- Documentazione online
- Test report formali (si veda il paragrafo 5.3.2)

È possibile utilizzare una o più di queste opzioni. Una comunicazione più formale può essere più appropriata per i team distribuiti, dove la comunicazione diretta face-to-face non è sempre possibile, a causa della distanza geografica o delle differenze di fuso orario. In genere, stakeholder differenti sono interessati a differenti tipi di informazione, quindi dovrebbe essere fatto un tailoring della comunicazione.

5.4. Configuration Management

Nel testing, il Configuration Management (CM) fornisce una disciplina per l'identificazione, il controllo e il monitoraggio, come elementi (item) di configurazione, dei prodotti di lavoro come test plan, strategie di test, condizioni di test, test case, test script, risultati dei test, test log e test report.

Per un configuration item complesso (es. un ambiente di test), il configuration management registra i configuration item che lo compongono, le loro relazioni e versioni. Se il configuration item viene approvato

per il testing, diventa una baseline e può essere modificato solo attraverso un processo formale di controllo delle modifiche (change control).

Il configuration management mantiene un record dei configuration item modificati quando viene creata una nuova baseline. È possibile tornare a una baseline precedente per riprodurre i risultati dei test precedenti.

Per supportare adeguatamente il testing, il configuration management garantisce quanto segue:

- Tutti i configuration item, compresi gli elementi del test (singole parti dell'oggetto di test), sono identificati in modo univoco, controllati nella versione, tracciati per le modifiche, e correlati ad altri configuration item in modo da mantenere la tracciabilità durante l'intero processo di test
- Tutti gli elementi di documentazione e gli elementi software identificati sono referenziati senza ambiguità nella documentazione di test

Continuous Integration, Continuous Delivery, Continuous Deployment e il testing associato sono generalmente implementati come parte di una pipeline automatizzata DevOps (si veda il paragrafo 2.1.4), in cui è normalmente incluso il configuration management automatizzato.

5.5. Defect Management

Poiché uno dei principali obiettivi del testing è trovare i difetti, un processo di defect management è essenziale. Si fa riferimento ai “difetti", anche se le anomalie segnalate possono rivelarsi veri e propri difetti oppure qualcos'altro (ad esempio, un falso positivo, una change request). Questo viene risolto durante il processo di gestione dei defect report. Le anomalie possono essere segnalate in qualsiasi fase del ciclo di vita dello sviluppo software, in una forma che dipende dal ciclo di vita. Come minimo, il processo di defect management include un workflow per la gestione delle singole anomalie, dalla loro scoperta alla loro chiusura, e le regole per la loro classificazione. Il workflow comprende tipicamente le attività per il log delle anomalie segnalate, la loro analisi e classificazione, la decisione sul fixing del bug individuato o lasciarlo nello stato in cui si trova, e infine la chiusura del defect report. Il processo deve essere seguito da tutti gli stakeholder coinvolti. È consigliabile gestire i difetti derivanti dal testing statico (in particolare l'analisi statica) in modo similare.

I defect report tipici hanno i seguenti obiettivi:

- Fornire le informazioni sufficienti ai responsabili della gestione e della risoluzione dei difetti segnalati, per poter risolvere il problema
- Fornire un mezzo per monitorare la qualità del prodotto di lavoro
- Fornire idee per il miglioramento del processo di sviluppo e test

Un defect report creato durante il testing dinamico di solito include:

- Un identificativo univoco
- Il titolo con un breve riassunto dell'anomalia segnalata
- La data di apertura del defect report, l’organizzazione, l’autore e il ruolo di chi ha segnalato il difetto
- L’identificativo dell'oggetto di test e dell'ambiente di test

- Il contesto del difetto (es. il test case che ha rilevato l'anomalia, l'attività di test e la fase del ciclo di vita dello sviluppo software in corso, e altre informazioni rilevanti come la tecnica di test, la checklist o i dati di test utilizzati)
- Una descrizione del failure per consentirne la riproduzione e la risoluzione, compresi i passi che hanno rilevato l'anomalia e i test log rilevanti, dump del database, screenshot o registrazioni pertinenti
- Risultato atteso e risultato effettivo
- Severità del difetto (grado di impatto) sugli interessi degli stakeholder o sui requisiti
- Priorità della correzione
- Stato del difetto (es. aperto, differito, duplicato, in attesa di risoluzione, in attesa di testing confermativo, riaperto, chiuso, rifiutato)
- Riferimenti (es. il test case la cui esecuzione ha rilevato l'anomalia)

Alcuni di questi dati possono essere inclusi automaticamente quando si utilizzano strumenti di defect management (es. identificativo, data, autore e stato iniziale). I template di documento per un defect report e un esempio di defect report possono essere trovati nello standard ISO/IEC/IEEE 29119-3, che fa riferimento ai defect report come incident report.

6. Strumenti di Test - 20 minuti

Parole chiave

test automation

Obiettivi di Apprendimento per il Capitolo 6:

6.1 Strumenti di Supporto per il Testing

FL-6.1.1 (K2) Spiegare come differenti tipi di strumenti di test supportano il testing

6.2 Benefici e Rischi della Test Automation

FL-6.2.1 (K1) Richiamare i benefici e i rischi della test automation

6.1. Strumenti di Supporto per il Testing

Gli strumenti di test supportano e facilitano molte attività di test. Esempi includono, ma non si limitano a:

- **Strumenti di management** - Aumentano l'efficienza del processo di test facilitando la gestione del ciclo di vita dello sviluppo software, il requirements management, il test management, il defect management, il configuration management
- **Strumenti di testing statico** - Supportano il tester nell'esecuzione delle review e dell'analisi statica
- **Strumenti per la progettazione e l'implementazione dei test** - Facilitano la generazione dei test case, dei dati di test e delle procedure di test
- **Strumenti di esecuzione dei test e di copertura** - Facilitano l'esecuzione automatizzata dei test e la misura della copertura
- **Strumenti per il testing non funzionale** - Permettono ai tester di eseguire il testing non funzionale che è difficile o impossibile eseguire manualmente
- **Strumenti DevOps** - Supportano la pipeline di delivery DevOps, il tracciamento del workflow, i processi di build automatizzati, la Continuous Integration/Continuous Delivery (CI/CD)
- **Strumenti di collaborazione** - Facilitano la comunicazione
- **Strumenti che supportano la scalabilità e la standardizzazione del rilascio** (es. macchine virtuali, strumenti per i container)
- Qualsiasi altro **strumento di supporto al testing** (es. un foglio di calcolo è uno strumento di test nel contesto del testing).

6.2. Benefici e Rischi della Test Automation

La semplice acquisizione di uno strumento non garantisce il successo. Ogni nuovo strumento richiederà un effort per ottenere benefici reali e duraturi (ad esempio per l'introduzione, la manutenzione e la formazione sullo strumento). Esistono anche alcuni rischi, che devono essere analizzati e mitigati.

I potenziali benefici legati all'utilizzo della test automation includono:

- Risparmio di tempo grazie alla riduzione del lavoro manuale ripetitivo (es. esecuzione dei regression test, reinserimento degli stessi dati di test, confronto dei risultati attesi rispetto ai risultati effettivi, e verifica degli standard di codifica)
- Prevenzione di semplici errori umani attraverso una maggiore consistenza e ripetibilità (es. i test sono derivati in modo consistente dai requisiti, i dati di test vengono creati in modo sistematico e i test vengono eseguiti da uno strumento nello stesso ordine e con la stessa frequenza)
- Valutazione più oggettiva (es. copertura) e generazione di misure troppo complicate per essere derivate dall'uomo
- Accesso più facile alle informazioni sul testing per supportare il test management e la reportistica dei test (es. statistiche, grafici e dati aggregati sull'avanzamento dei test, tassi di difettosità e durata dell'esecuzione dei test)

- Riduzione dei tempi di esecuzione dei test che permette di fornire una rilevazione anticipata dei difetti, un feedback più rapido e un time-to-market più veloce
- Aumento del tempo che i tester possono dedicare alla progettazione di test nuovi, più approfonditi e più efficaci

I potenziali rischi legati all'utilizzo della test automation includono:

- Aspettative irrealistiche sui benefici di uno strumento (incluse le funzionalità e la facilità d'uso)
- Stime non accurate dei tempi, costi ed effort necessari per introdurre uno strumento, per manutenere i test script e per modificare il processo di test manuale esistente
- Utilizzo di uno strumento di test quando è più appropriato il testing manuale
- Troppa fiducia nello strumento, ignorando ad esempio la necessità del pensiero critico umano
- Dipendenza dal fornitore dello strumento, che può cessare l'attività, ritirare lo strumento, venderlo a un fornitore differente o fornire un'assistenza insufficiente (es. risposte alle domande, aggiornamenti e correzioni di difetti)
- Sospensione di un software open-source, che causa la non disponibilità di ulteriori aggiornamenti
- Frequenti aggiornamenti come ulteriori sviluppi di componenti interni di un software open-source
- Non compatibilità dello strumento di test automation con la piattaforma di sviluppo
- Scelta di uno strumento non idoneo e non conforme ai requisiti normativi e/o agli standard safety.

7. Riferimenti

Standard

ISO/IEC/IEEE 29119-1 (2022) Software and systems engineering – Software testing – Part 1: General Concepts

ISO/IEC/IEEE 29119-2 (2021) Software and systems engineering – Software testing – Part 2: Test processes

ISO/IEC/IEEE 29119-3 (2021) Software and systems engineering – Software testing – Part 3: Test documentation

ISO/IEC/IEEE 29119-4 (2021) 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

ISO/IEC/IEEE 14764:2022 – Software engineering – Software life cycle processes – Maintenance

ISO 31000 (2018) Risk management – Principles and guidelines

Libri

Adzic, G. (2009) Bridging the Communication Gap: Specification by Example and Agile Acceptance Testing, Neuri Limited

Ammann, P. and Offutt, J. (2016) Introduction to Software Testing (2e), Cambridge University Press

Andrews, M. and Whittaker, J. (2006) How to Break Web Software: Functional and Security Testing of Web Applications and Web Services, Addison-Wesley Professional

Beck, K. (2003) Test Driven Development: By Example, Addison-Wesley

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

Boehm, B. (1981) Software Engineering Economics, Prentice Hall, Englewood Cliffs, NJ

Buxton, J.N. and Randell B., eds (1970), Software Engineering Techniques. Report on a conference sponsored by the NATO Science Committee, Rome, Italy, 27–31 October 1969, p. 16

Chelimsky, D. et al. (2010) The Rspec Book: Behaviour Driven Development with Rspec, Cucumber, and Friends, The Pragmatic Bookshelf: Raleigh, NC

Cohn, M. (2009) Succeeding with Agile: Software Development Using Scrum, Addison-Wesley

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: A Practical Guide for Testers and Agile Teams, Pearson Education: Boston MA

Forgács, I., and Kovács, A. (2019) Practical Test Design: Selection of traditional and automated test design techniques, BCS, The Chartered Institute for IT

Gawande A. (2009) The Checklist Manifesto: How to Get Things Right, New York, NY: Metropolitan Books

Gärtner, M. (2011), ATDD by Example: A Practical Guide to Acceptance Test-Driven Development, Pearson Education: Boston MA

Gilb, T., Graham, D. (1993) Software Inspection, Addison Wesley

Hendrickson, E. (2013) Explore It!: Reduce Risk and Increase Confidence with Exploratory Testing, The Pragmatic Programmers

Hetzel, B. (1988) The Complete Guide to Software Testing, 2nd ed., John Wiley and Sons

Jeffries, R., Anderson, A., Hendrickson, C. (2000) Extreme Programming Installed, Addison-Wesley Professional

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

Kan, S. (2003) Metrics and Models in Software Quality Engineering, 2nd ed., Addison-Wesley

Kaner, C., Falk, J., and Nguyen, H.Q. (1999) Testing Computer Software, 2nd ed., Wiley

Kaner, C., Bach, J., and Pettichord, B. (2011) Lessons Learned in Software Testing: A Context-Driven Approach, 1st ed., Wiley

Kim, G., Humble, J., Debois, P. and Willis, J. (2016) The DevOps Handbook, Portland, OR

Koomen, T., van der Aalst, L., Broekman, B. and Vroon, M. (2006) TMap Next for result-driven testing, UTN Publishers, The Netherlands

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

O'Regan, G. (2019) Concise Guide to Software Testing, Springer Nature Switzerland

Pressman, R.S. (2019) Software Engineering. A Practitioner's Approach, 9th ed., McGraw Hill

Roman, A. (2018) Thinking-Driven Testing. The Most Reasonable Approach to Quality Control, Springer Nature Switzerland

Van Veenendaal, E (ed.) (2012) Practical Risk-Based Testing, The PRISMA Approach, UTN Publishers: The Netherlands

Watson, A.H., Wallace, D.R. and McCabe, T.J. (1996) Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric, U.S. Dept. of Commerce, Technology Administration, NIST

Westfall, L. (2009) The Certified Software Quality Engineer Handbook, ASQ Quality Press

Whittaker, J. (2002) How to Break Software: A Practical Guide to Testing, Pearson

Whittaker, J. (2009) Exploratory Software Testing: Tips, Tricks, Tours, and Techniques to Guide Test Design, Addison Wesley

Whittaker, J. and Thompson, H. (2003) How to Break Software Security, Addison Wesley

Wiegers, K. (2001) Peer Reviews in Software: A Practical Guide, Addison-Wesley Professional

Articoli e Pagine Web

Brykczynski, B. (1999) “A survey of software inspection checklists,” *ACM SIGSOFT Software Engineering Notes, 24(1), pp. 82-89*

Enders, A. (1975) “An Analysis of Errors and Their Causes in System Programs,” *IEEE Transactions on Software Engineering 1(2), pp. 140-149*

Manna, Z., Waldinger, R. (1978) “The logic of computer programming,” *IEEE Transactions on Software Engineering 4(3), pp. 199-229*

Marick, B. (2003) Exploration through Example, http://www.exampler.com/old-blog/2003/08/21.1.html#agile-testing-project-1

Nielsen, J. (1994) “Enhancing the explanatory power of usability heuristics,” *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Celebrating Interdependence, ACM Press, pp. 152–158*

Salman. I. (2016) “Cognitive biases in software quality and testing,” *Proceedings of the 38th International Conference on Software Engineering Companion (ICSE '16), ACM, pp. 823-826.*

Wake, B. (2003) “INVEST in Good Stories, and SMART Tasks,” https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/

8. Appendice A - Obiettivi di Apprendimento/Livello Cognitivo della Conoscenza

I seguenti obiettivi di apprendimento sono definiti come applicabili a questo Syllabus. Ogni argomento del Syllabus sarà esaminato in base al relativo obiettivo di apprendimento. Gli obiettivi di apprendimento iniziano con un verbo corrispondente al livello cognitivo della conoscenza, come elencato di seguito.

Livello 1: Ricordare (K1) - Il candidato è in grado di ricordare, riconoscere e richiamare un termine o un concetto.

Verbi: identificare, richiamare (alla memoria), ricordare, riconoscere.

Esempi:

- "Identificare gli obiettivi tipici del test".
- "Richiamare i concetti della piramide di test".
- "Riconoscere come un tester aggiunge valore alla pianificazione dell'iterazione (iteration planning) e alla pianificazione della release (release planning)".

Livello 2: Comprendere (K2) - Il candidato è in grado di selezionare le ragioni o le spiegazioni delle affermazioni relative all'argomento, ed è in grado di riassumere, confrontare, classificare e fornire esempi per il concetto del testing.

Verbi: classificare, confrontare, comparare, contrastare, mettere in contrapposizione, differenziare, distinguere, esemplificare, fornire esempi, spiegare, fornire esempi, interpretare, riassumere.

Esempi:

- "Classificare le differenti opzioni per la scrittura dei criteri di accettazione".
- "Confrontate i differenti ruoli nel testing" (cercare le somiglianze, le differenze o entrambe).
- "Distinguere tra rischi di progetto e rischi di prodotto" (essere in grado di differenziare i concetti).
- "Fornire esempi sullo scopo e sul contenuto di un test plan".
- "Spiegare l'impatto del contesto sul processo di test".
- "Riassumere le attività del processo di review".

Livello 3: Applicare (K3) - Il candidato è in grado di eseguire una procedura quando si trova di fronte a un task che gli è familiare, o è in grado di selezionare la procedura corretta e applicarla nell'ambito di un determinato contesto.

Verbi: applicare (una procedura), implementare, preparare, utilizzare.

Esempi:

- "Applicare la prioritizzazione dei test case" (dovrebbe far riferimento a una procedura, tecnica, processo, algoritmo, ecc.)
- "Preparare un defect report".
- "Utilizzare l'analisi ai valori limite per derivare i test case".

Riferimenti per i livelli cognitivi degli obiettivi di apprendimento:

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

9. Appendice B - Matrice di Tracciabilità dei Risultati di Business rispetto agli Obiettivi di Apprendimento

Questo capitolo elenca il numero di Obiettivi di Apprendimento Foundation Level relativi ai Risultati di business e la tracciabilità tra i Risultati di Business Foundation Level e gli Obiettivi di Apprendimento Foundation Level.

Risultati di Business: Foundation Level		FL-BO1	FL-BO2	FL-BO3	FL-BO4	FL-BO5	FL-BO6	FL-BO7	FL-BO8	FL-BO9	FL-BO10	FL-BO11	FL-BO12	FL-BO13	FL-BO14
BO1	Comprendere che cos'è il testing e perché è vantaggioso	6													
BO2	Comprendere i concetti fondamentali del testing del software		22												
BO3	Identificare l'approccio del test e le attività di test da implementare in base al contesto del testing			6											
BO4	Valutare e migliorare la qualità della documentazione				9										
BO5	Aumentare l'efficacia e l'efficienza del testing					20									
BO6	Allineare il processo di test con il ciclo di vita dello sviluppo software						6								
BO7	Comprendere i principi di test management							6							
BO8	Scrivere e comunicare defect report chiari e comprensibili								1						
BO9	Comprendere i fattori che influenzano le priorità e gli effort relativi al testing									7					
BO10	Lavorare come parte di un team cross-funzionale										8				
BO11	Conoscere i rischi e i benefici legati alla test automation											1			
BO12	Identificare le competenze fondamentali richieste per il testing												5		
BO13	Comprendere l'impatto del rischio sul testing													4	
BO14	Comunicare efficacemente sui progressi e sulla qualità del test														4

Capitolo/ paragrafo/ sotto-paragrafo	Obiettivo di Apprendimento	Livello-K	RISULTATI DI BUSINESS														
			FL-BO1	FL-BO2	FL-BO3	FL-BO4	FL-BO5	FL-BO6	FL-BO7	FL-BO8	FL-BO9	FL-BO10	FL-BO11	FL-BO12	FL-BO13	FL-BO14
Capitolo 1	**Fondamenti del Testing**															
1.1	**Cos'è il Testing?**															
1.1.1	Identificare gli obiettivi tipici del testing	K1	X													
1.1.2	Differenziare il testing dal debugging	K2		X												
1.2	**Perché il Testing è Necessario?**															
1.2.1	Fornire esempi del perché il testing è necessario	K2	X													
1.2.2	Richiamare la relazione tra testing e quality assurance	K1		X												
1.2.3	Distinguere tra root cause, errore, difetto e failure	K2		X												
1.3	**Principi del Testing**															
1.3.1	Spiegare i sette principi del testing	K2		X												
1.4	**Attività di Test, Testware e Ruoli del Test**															
1.4.1	Riassumere le differenti attività di test e i compiti del test	K2			X											
1.4.2	Spiegare l'impatto del contesto sul processo di test	K2			X			X								
1.4.3	Differenziare il testware che supporta le attività di test	K2			X											

1.4.4	Spiegare il valore di mantenere la tracciabilità	K2				X	X									
1.4.5	Confrontare i differenti ruoli nel testing	K2										X				
1.5	**Competenze Fondamentali e Buone Pratiche nel Testing**															
1.5.1	Fornire esempi delle competenze generiche richieste per il testing	K2												X		
1.5.2	Richiamare i vantaggi dell'approccio whole-team	K1										X				
1.5.3	Distinguere i benefici e gli svantaggi dell'indipendenza del testing	K2			X											
Capitolo 2	**Il Testing all'interno del Ciclo di Vita dello Sviluppo Software**															
2.1	**Il Testing nel Contesto di un Ciclo di Vita dello Sviluppo Software**															
2.1.1	Spiegare l'impatto sul testing del ciclo di vita dello sviluppo software selezionato	K2						X								
2.1.2	Ricordare le buone pratiche del testing che si applicano a tutti i cicli di vita dello sviluppo software	K1						X								
2.1.3	Ricordare gli esempi di approcci allo sviluppo test-first	K1					X									
2.1.4	Riassumere come DevOps potrebbe avere un impatto sul testing	K2					X	X			X	X				
2.1.5	Spiegare l'approccio shift-left	K2					X	X								
2.1.6	Spiegare come le retrospettive possono essere utilizzate come meccanismo per il process improvement (miglioramento del processo)	K2					X					X				
2.2	**Livelli di Test e Tipi di Test**															
2.2.1	Distinguere i differenti livelli di test	K2		X	X											
2.2.2	Distinguere i differenti tipi di test	K2		X												
2.2.3	Distinguere il testing confermativo dal regression testing	K2		X												
2.3	**Testing di Manutenzione**															

2.3.1	Riassumere il testing di manutenzione e i suoi trigger di attivazione	K2		X					X							
Capitolo 3	**Testing Statico**															
3.1	**Fondamenti del Testing Statico**															
3.1.1	Riconoscere i tipi di prodotti che possono essere esaminati con le diverse tecniche di test statico	K1				X	X									
3.1.2	Spiegare il valore del testing statico	K2	X			X	X									
3.1.3	Confrontare e mettere in contrapposizione il testing statico e dinamico	K2				X	X									
3.2	**Feedback e Processo di Feedback**															
3.2.1	Identificare i benefici di un feedback anticipato e frequente da parte degli stakeholder	K1	X			X						X				
3.2.2	Riassumere le attività del processo di review	K2			X	X										
3.2.3	Ricordare le responsabilità assegnate ai ruoli principali durante l'esecuzione delle review	K1				X								X		
3.2.4	Confrontare e mettere in contrapposizione i diversi tipi di review	K2		X												
3.2.5	Ricordare i fattori che contribuiscono al successo di una review	K1					X							X		
Capitolo 4	**Analisi e Progettazione dei Test**															
4.1	**Panoramica sulle Tecniche di Test**															
4.1.1	Distinguere le tecniche di test black-box, white-box e basate sull'esperienza	K2		X												
4.2	**Tecniche di Test Black-box**															
4.2.1	Utilizzare il partizionamento di equivalenza per derivare i test case	K3					X									
4.2.2	Utilizzare l'analisi ai valori limite per derivare i test case	K3					X									
4.2.3	Utilizzare il testing delle transizioni di stato per derivare i test case	K3					X									

4.2.4	Utilizzare il testing delle transizioni di stato per derivare i test case	K3					X									
4.3	**Tecniche di Test White-box**															
4.3.1	Spiegare il testing delle istruzioni	K2		X												
4.3.2	Spiegare il testing dei rami	K2		X												
4.3.3	Spiegare il valore del testing white-box	K2	X	X												
4.4	**Tecniche di Test basate sull'Esperienza**															
4.4.1	Spiegare l'error guessing	K2		X												
4.4.2	Spiegare il testing esplorativo	K2		X												
4.4.3	Spiegare il testing checklist-based	K2		X												
4.5	**Approcci del Test basati sulla Collaborazione**															
4.5.1	Spiegare come scrivere le user story in collaborazione con sviluppatori e i rappresentanti di business	K2				X						X				
4.5.2	Classificare le differenti opzioni per la scrittura dei criteri di accettazione	K2										X				
4.5.3	Utilizzare l'acceptance test-driven development (ATDD) per derivare i test case	K3					X									
Capitolo 5	**Gestione delle Attività di Test**															
5.1	**Pianificazione dei Test**															
5.1.1	Fornire esempi sullo scopo e sul contenuto di un test plan	K2		X					X							
5.1.2	Riconoscere come un tester aggiunge valore alla pianificazione dell'iterazione (iteration planning) alla pianificazione della release (release planning)	K1	X									X		X		
5.1.3	Confrontare e mettere in contrapposizione i criteri di ingresso e i criteri di uscita	K2				X		X								X
5.1.4	Utilizzare le tecniche di stima per calcolare l'effort del test richiesto	K3							X		X					

5.1.5	Applicare la prioritizzazione dei test case	K3							X		X					
5.1.6	Richiamare i concetti della piramide di test	K1		X												
5.1.7	Riassumere i quadranti del testing e le loro relazioni con i livelli di test e i tipi di test	K2		X							X					
5.2	**Risk Management**															
5.2.1	Identificare il livello di rischio utilizzando la probabilità del rischio e l'impatto del rischio	K1							X						X	
5.2.2	Distinguere tra rischi di progetto e rischi di prodotto	K2		X											X	
5.2.3	Spiegare come l'analisi del rischio del prodotto possa influenzare la completezza e l'ambito del testing	K2					X				X				X	
5.2.4	Spiegare quali misure possono essere adottate in risposta ai rischi di prodotto analizzati	K2		X			X								X	
5.3	**Monitoraggio dei Test, Controllo dei Test e Completamento dei Test**															
5.3.1	Richiamare le metriche utilizzate per il testing	K1									X					X
5.3.2	Riassumere gli scopi, i contenuti e i destinatari dei test report	K2					X				X					X
5.3.3	Fornire esempi su come comunicare lo stato di avanzamento del testing	K2												X		X
5.4	**Configuration Management**															
5.4.1	Riassumere come il configuration management supporta il testing	K2					X		X							
5.5	**Defect Management**															
5.5.1	Preparare un defect report	K3		X						X						
Capitolo 6	**Strumenti di Test**															
6.1	**Strumenti di Supporto per il Testing**															
6.1.1	Spiegare come differenti tipi di strumenti di test supportano il testing	K2					X									

6.2	**Benefici e Rischi della Test Automation**															
6.2.1	Richiamare i benefici e i rischi della test automation	K1					X						X			

10. Appendice C – Release Note

Il Syllabus ISTQB® Foundation Level v4.0 è un aggiornamento importante (major release) basato sul Syllabus Foundation Level v3.1.1 e sul Syllabus Agile Tester 2014. Per questo motivo, non sono presenti release note dettagliate per capitolo e paragrafo. Tuttavia, di seguito viene fornito un riepilogo delle principali modifiche. Inoltre, in un documento separato di Release Note, ISTQB® fornisce la tracciabilità tra gli Obiettivi di Apprendimento (LO, Learning Objective) del nuovo Syllabus Foundation Level v4.0 e gli Obiettivi di Apprendimento della versione 3.1.1 del Syllabus Foundation Level e della versione 2014 del Syllabus Agile Tester, evidenziando quali Obiettivi di Apprendimento sono stati aggiunti, aggiornati o eliminati.

Al momento della stesura del Syllabus (2022-2023) più di un milione di persone in oltre 100 paesi hanno sostenuto l'esame Foundation Level e più di 800.000 sono i tester certificati a livello mondiale. Con l'aspettativa che tutti abbiano letto il Syllabus Foundation Level per essere in grado di superare l'esame, questo rende il Syllabus Foundation probabilmente il documento del testing del software più letto di sempre! Questo importante aggiornamento è stato realizzato nel rispetto di questo patrimonio e per migliorare il valore per altre centinaia di migliaia di persone sul livello di qualità che ISTQB® offre alla comunità globale del testing.

In questa versione tutti i LO sono stati modificati per renderli atomici, e per creare una tracciabilità uno-a-uno tra i LO e i paragrafi del Syllabus, in modo da non avere un contenuto senza avere anche un LO. L'obiettivo è rendere questa versione più facile da leggere, comprendere, imparare e tradurre, focalizzandosi sull'aumento dell'utilità pratica e sull'equilibrio tra conoscenze e competenze.

Questa major release ha apportato le seguenti modifiche:

- Riduzione della dimensione complessiva del Syllabus. Il Syllabus non è un libro di testo, ma un documento che serve a delineare gli elementi di base di un corso introduttivo sul testing del software, includendo quali argomenti dovrebbero essere coperti e a quale livello. In particolare:
    - Nella maggior parte dei casi gli esempi sono esclusi dal testo. È compito di un Training Provider fornire gli esempi e gli esercizi durante la formazione
    - È stata seguita la "Checklist per la stesura del Syllabus", che suggerisce la dimensione massima del testo per i LO a ciascun livello K (K1 = max. 10 righe, K2 = max. 15 righe, K3 = max. 25 righe).
- Riduzione del numero di LO rispetto ai Syllabi Foundation Level (FL) v3.1.1 e Agile Tester (AT) v2014.
    - 14 LO Livello K1 rispetto ai 21 LO in FL v3.1.1 (15) e AT 2014 (6)
    - 42 LO Livello K2 rispetto ai 53 LO in FL v3.1.1 (40) e AT 2014 (13)
    - 8 LO Livello K3 rispetto ai 15 LO in FL v3.1.1 (7) e AT 2014 (8)
- Vengono forniti riferimenti più ampi a libri e articoli classici e/o autorevoli sul testing del software e su argomenti correlati.
- Principali modifiche al Capitolo 1 (Fondamenti del Testing)
    - Il paragrafo sulle competenze del testing è stato ampliato e migliorato

- È stato aggiunto un paragrafo sull'approccio whole-team (K1)
- Il paragrafo sull'indipendenza del testing è stato spostato dal Capitolo 5 al Capitolo 1

- Principali modifiche al Capitolo 2 (Il Testing all'interno del Ciclo di Vita dello Sviluppo Software)
  - I paragrafi 2.1.1 e 2.1.2 sono stati riscritti e migliorati, i LO corrispondenti sono stati modificati
  - Maggiore focus ad alcune pratiche come: approccio test-first (K1), shift-left (K2), retrospettive (K2)
  - È stato inserito un nuovo paragrafo sul testing nel contesto DevOps (K2)
  - Il livello del testing di integrazione è stato suddiviso in due livelli di test separati: testing di integrazione dei componenti e testing di integrazione dei sistemi
- Principali modifiche al Capitolo 3 (Testing Statico)
  - È stato eliminato il paragrafo sulle tecniche di review, insieme al relativo LO Livello K3 (applicare una tecnica di review)
- Principali modifiche al Capitolo 4 (Analisi e Progettazione dei Test)
  - Il testing degli use case è stato cancellato (ma ancora presente nel Syllabus Advanced Level Test Analyst)
  - Maggior focus all'approccio al testing basato sulla collaborazione: nuovo LO Livello K3 sull'utilizzo di ATDD per derivare i test case e due nuovi LO Livello K2 sulle user story e i criteri di accettazione
  - Il testing e la copertura delle decisioni sono stati sostituiti con il testing e la copertura dei rami (in primo luogo, la copertura dei rami è più comunemente utilizzata nella pratica; in secondo luogo, diversi standard definiscono la "decisione" in modo differente rispetto al "ramo"; in terzo luogo, questo risolve un sottile, ma grave difetto del vecchio Syllabus Foundation Level FL2018 che sostiene che "il 100% di copertura delle decisioni implica il 100% di copertura delle istruzioni" - questa frase non è vera nel caso di codice senza decisioni)
  - Il paragrafo sul valore del testing white-box è stato migliorato
- Principali modifiche al Capitolo 5 (Gestione delle Attività di Test)
  - Il paragrafo sulle strategie/approcci del test è stato eliminato
  - È stato aggiunto un nuovo LO Livello K3 sulle tecniche di stima per stimare l'effort del test
  - Maggiore focus ai ben noti concetti e strumenti Agile nel test management: pianificazione dell'iterazione (iteration planning) e pianificazione della release (release planning) (K1), piramide di test (K1) e quadranti del testing (K2)
  - Il paragrafo sul risk management è stato strutturato meglio, descrivendo le quattro attività principali: identificazione del rischio, valutazione del rischio, mitigazione del rischio e monitoraggio del rischio
- Principali modifiche al Capitolo 6 (Strumenti di Test)

- I contenuti su alcuni temi della test automation sono stati ridotti in quanto troppo avanzati per il Foundation Level – è stato eliminato il paragrafo sulla selezione degli strumenti, sull'esecuzione di progetti pilota e sull'introduzione degli strumenti nell'organizzazione

11. Indice