

Certified Tester Advanced Level Agile Tester (CTAL-AT) Syllabus

Versione 2.0

International Software Testing Qualifications Board



Avviso di Copyright

Avviso di Copyright © International Software Testing Qualifications Board (di seguito denominato ISTQB®).

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

Copyright © 2026, gli autori della versione 2.0: Wim Decoutere, Michaël Pilaeten (Product Owner), Adam Roman, Murian Song, Nele Van Asch.

Copyright © 2014, gli autori: Rex Black (Chair), Anders Claesson, Gerry Coleman, Bertrand Cornanguer (Vice Chair), Istvan Forgacs, Alon Linetzki, Tilo Linz, Leo van der Aalst, Marie Walsh e Stephan Weber.

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

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

Cronologia delle revisioni

Versione	Data	Osservazioni
CT-AT v1.0 (2014)	2014-05-31	Prima versione (sostituita)
CTAL-AT v2.0 (2026)	2026-04-17	Rielaborazione e riposizionamento

Indice dei contenuti

Avviso di Copyright.....	2
Cronologia delle revisioni	3
Indice dei contenuti	4
Ringraziamenti	7
0 Introduzione.....	9
0.1 Scopo di questo Syllabus	9
0.2 Il Certified Tester Agile Tester nel Testing del Software	9
0.3 Percorso di Carriera per i Tester	9
0.4 Business Outcome	9
0.5 Obiettivi di Apprendimento e Livello Cognitivo di Conoscenza	10
0.6 L'Esame per il Certificato CTAL - Agile Tester	11
0.7 Accredитamento	11
0.8 Gestione degli Standard	11
0.9 Livello di Dettaglio	11
0.10 Come è Organizzato questo Syllabus	12
1 Sfide relative alla Strategia di Test e all'Approccio del Test - 60 minuti	13
Introduzione all'Agile	14
1.1 Tipi di Test	16
1.2 Testing End-to-End	17
1.3 Testing Formale e Testing Olistico	18
1.4 Approcci del Regression Testing	20
2 Persone e Team - 60 minuti	21
Introduzione.....	22
2.1 Whole Team Approach	22
2.1.1 Generalizzazione e specializzazione all'interno di un team	22
2.1.2 Motivare i rappresentanti del business a partecipare al testing	23
2.1.3 Supportare gli sviluppatori	24
2.2 Tissue Tester	24
3 Test Management e Test Process Improvement - 210 minuti	26
Introduzione.....	27

3.1	Pianificazione dei Test	28
3.1.1	Pianificazione dei test nello sviluppo software Agile	28
3.1.2	Strategia di test del progetto.....	29
3.2	Monitoraggio dei Test e Controllo dei Test	30
3.3	Reporting del Test.....	31
3.4	Test Process Improvement	32
3.4.1	Metriche per il Test Process Improvement	32
3.4.2	Test Process Improvement nello sviluppo software Agile	33
4	Shift-Left - 135 minuti	35
	Introduzione.....	36
4.1	Utilizzare lo Shift-Left per Migliorare la Qualità della Base di Test.....	36
4.1.1	Testware come requisiti.....	36
4.1.2	Storyboarding e Testboarding	37
4.1.3	Example Mapping	37
4.1.4	Bias nel testing Agile	38
4.1.5	User Story Slicing	39
4.2	Shift-Left e Requirements Engineering	40
5	Approcci e Tecniche di Test Agile - 285 minuti	42
	Introduzione.....	43
5.1	Testing Esplorativo	43
5.1.1	Euristiche di test	43
5.1.2	Mnemonici di test.....	44
5.1.3	Tour di test.....	44
5.1.4	Creazione delle test charter.....	45
5.1.5	Eseguire il testing esplorativo.....	47
5.2	Testing Assistito	48
5.2.1	Mob testing	48
5.2.2	Pair testing.....	49
5.2.3	Vibe testing	50
5.3	Test Smell	50
6	Test Automation e Strumenti di Test - 30 minuti	54
	Introduzione.....	55

6.1	Test Automation nello Sviluppo Software Agile	55
6.2	Strumenti di Test nello Sviluppo Software Agile	56
7	Riferimenti	58
8	Ulteriori Letture.....	61
9	Appendice A – Elenco delle Abbreviazioni	62
10	Appendice B – Termini Specifici del Dominio	63
11	Appendice C – Obiettivi di Apprendimento/Livello Cognitivo di Conoscenza	65
12	Appendice D – Matrice di Tracciabilità dei Business Outcome agli Obiettivi di Apprendimento	68
13	Appendice E – Release Note	76
14	Marchi Registrati	77
15	Indice	78

Ringraziamenti

Questo documento è stato pubblicato formalmente dalla General Assembly di ISTQB® il 17 aprile 2026. È stato redatto da un team di International Software Testing Qualifications Board: Wim Decoutere, Michaël Pilaeten, Adam Roman, Murian Song, Nele Van Asch.

Il team ringrazia il Dr. Stuart Reid e Mark Rutz per la review tecnica, nonché il team di review e i Member Board per i loro suggerimenti e contributi.

Le seguenti persone hanno partecipato alla revisione, ai commenti e alla votazione di questo Syllabus: Mariam Abdellatif, Hatim Abuazab, Bíró Ádám, Tom Adams, Gergely Agnecz, Laura Albert, Tobias Anderson, Chris Van Bael, André Baumann, Jürgen Beniermann, Menno van den Berg, Ruud van Berkum, Armin Born, Earl Burba, Simeone Chiumarulo, Marco Ciarlito, Alessandro Collino, Walter Eder, Michael Fischlein, Valeriya Gagarina, Attila Gyuri, Ferenc Hamori, Andrea Horvath, Sagar Joshi, Mattijs Kemmink, Jędrzej Kwapiński, Thomas Letzkus, Ine Lutterman, Niranjan Maharajh, Judy McKay, Imre Meszaros, Gary Mogyorodi, Thaer Mustafa, Frank Neiryneck, Andreas Neumeister, Melanie Neumueller, Ingvar Nordström, Tauhida Parveen, Tal Pe'er, Christoph Peters, Lukas Piska, Horst Pohlmann, Andrew Pollner, Nishan Portoyan, Meile Posthuma, Randall Rice, Jean François Riverin, Nicola De Rosa, Mark Rutz, Salvatore Sarno, Márton Siska, Klaus Skafte, Péter Sótér, Michael Stahl, Lucjan Stapp, Ioannis Symeonidis, Reifa Tangon, Richard Taylor, Benjamin Timmermans, Giancarlo Tomasig, Stephanie Ulrich, Csilla Varga, Linda Vreeswijk e Jana Zientková.

Il Foundation Level Working Group di International Software Testing Qualifications Board al momento del completamento del Syllabus Foundation Level Agile Extension v1.0 (2014): Rex Black (Chair), Bertrand Cornanguer (Vice Chair), Gerry Coleman (Learning Objectives Lead), Debra Friedenberg (Exam Lead), Alon Linetzki (Business Outcomes and Marketing Lead), Tauhida Parveen (Editor), e Leo van der Aalst (Development Lead).

Autori del Syllabus Foundation Level Agile Extension v1.0 (2014): Rex Black, Anders Claesson, Gerry Coleman, Bertrand Cornanguer, Istvan Forgacs, Alon Linetzki, Tilo Linz, Leo van der Aalst, Marie Walsh e Stephan Weber.

Revisori interni del Syllabus Foundation Level Agile Extension v1.0 (2014): Mette Bruhn-Pedersen, Christopher Clements, Alessandro Collino, Debra Friedenberg, Kari Kakkonen, Beata Karpinska, Sammy Kolluru, Jennifer Leger, Thomas Mueller, Tuula Pääkkönen, Meile Posthuma, Gabor Puhalla, Lloyd Roden, Marko Rytönen, Monika Stoecklein-Olsen, Robert Treffny, Chris Van Bael ed Erik van Veenendaal.

Il team ringrazia inoltre le seguenti persone, provenienti dai National Board e dalla comunità di esperti Agile, che hanno partecipato alla review, ai commenti e alla votazione del Syllabus Foundation Level Agile Extension v1.0 (2014): Dani Almog, Richard Berns, Stephen Bird, Monika Bögge, Afeng Chai, Josephine Crawford, Tibor Csöndes, Huba Demeter, Arnaud Foucal, Cyril Fumery, Kobi Halperin, Inga Hansen, Hanne Hinz, Jidong Hu, Phill Isles, Shirley Itah, Martin Klonk, Kjell Lauren, Igal Levi, Rik Marselis, Johan Meivert, Armin Metzger, Peter Morgan, Ninna Morin, Ingvar Nordstrom, Chris O'Dea, Klaus Olsen, Ismo Paukamainen, Nathalie Phung, Helmut Pichler, Salvatore Reale, Stuart Reid, Hans Rombouts, Petri Säilynoja, Soile Sainio, Lars-Erik Sandberg, Dakar Shalom, Jian Shen, Marco Sogliani, Lucjan Stapp, Yaron Tsubery, Sabine Uhde, Stephanie Ulrich, Tommi Välimäki, Jurian Van de Laar, Marnix Van den Ent, António Vieira Melo, Wenye Xu, Ester Zabar, Wenqiang Zheng, Peter Zimmerer, Stevan Zivanovic e Terry Zuo.

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

Revisione della traduzione italiana a cura di: Raffaele Bozzini, Salvatore Reale

0 Introduzione

0.1 Scopo di questo Syllabus

Questo Syllabus costituisce la base per l'International Software Testing Qualification di CTAL - Agile Tester. ISTQB® fornisce questo Syllabus come segue:

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

0.2 Il Certified Tester Agile Tester nel Testing del Software

La certificazione ISTQB® Certified Tester Agile Tester (CTAL-AT) è rivolta a chiunque sia coinvolto nel testing del software in progetti Agile. Ciò include persone che ricoprono ruoli come tester, test analyst, test engineer, test consultant, test manager, user acceptance tester e sviluppatori software. Questa certificazione è appropriata anche per chiunque desideri una migliore comprensione del testing del software Agile, inclusi product owner, Scrum master, quality manager, responsabili dello sviluppo software, business analyst, direttori IT e consulenti manageriali.

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 ampiezza e profondità di conoscenza. Chi consegue la certificazione ISTQB® CTAL - Agile Tester può essere interessato anche ad altri Syllabus specialistici, con particolare attenzione all'Agile Test Leadership at Scale. È possibile visitare il sito www.istqb.org per rimanere aggiornati sulle ultime informazioni relative allo schema di certificazione dei tester certificati ISTQB®.

0.4 Business Outcome

Questa sezione elenca i Business Outcome (Risultati di Business) previsti per un candidato che ha ottenuto la certificazione CTAL - Agile Tester.

Un Certified Agile Tester è in grado di:

Codice	Descrizione
CTAL-AT-BO1	Collaborare in team interfunzionali, avendo familiarità con i principi e le pratiche di base dello sviluppo software Agile
CTAL-AT-BO2	Adattare l'esperienza e le conoscenze di testing esistenti ai valori e ai principi Agile
CTAL-AT-BO3	Supportare il team Agile nella pianificazione dei test
CTAL-AT-BO4	Applicare approcci di sviluppo software Agile e tecniche di test rilevanti per garantire che i test forniscano una copertura adeguata
CTAL-AT-BO5	Assistere gli stakeholder di business nella definizione di user story, scenari, requisiti e criteri di accettazione comprensibili e testabili, secondo necessità
CTAL-AT-BO6	Creare e implementare vari approcci del test per lo sviluppo software Agile
CTAL-AT-BO7	Supportare e contribuire alla test automation in un progetto di sviluppo software Agile
CTAL-AT-BO8	Lavorare con gli altri membri del team e condividere informazioni con loro utilizzando stili e canali di comunicazione efficaci

0.5 Obiettivi di Apprendimento e Livello Cognitivo di Conoscenza

Gli Obiettivi di Apprendimento (LO, Learning Objective) supportano i Business Outcome e vengono utilizzati per creare gli esami di certificazione CTAL - Agile Tester.

In generale, tutti i contenuti di questo Syllabus sono oggetto d'esame, ad eccezione dell'Introduzione e delle Appendici. Le domande d'esame valuteranno la conoscenza delle parole chiave al livello K2 (vedi sotto) o degli obiettivi di apprendimento al rispettivo livello di conoscenza.

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

- K1: Ricordare (gli Obiettivi di Apprendimento K1 non sono presenti in questo Syllabus)
- K2: Comprendere
- K3: Applicare
- K4: Analizzare

Ulteriori dettagli ed esempi di obiettivi di apprendimento sono riportati nella Sezione 11.

Per tutti i termini elencati come parole chiave appena sotto i titoli dei capitoli, è necessario ricordare (K1) il nome corretto e la definizione dal Glossario ISTQB®, inoltre la definizione dal Glossario ISTQB® deve essere compresa (K2), anche se non esplicitamente menzionata nell'obiettivo di apprendimento.

0.6 L'Esame per il Certificato CTAL - Agile Tester

L'esame per il certificato CTAL - Agile Tester sarà basato su questo Syllabus. Le risposte alle domande d'esame possono richiedere l'uso di materiale basato su più di una sezione di questo Syllabus. Tutte le sezioni del Syllabus sono esaminabili, ad eccezione dell'Introduzione e delle Appendici. Gli standard e i libri sono inclusi come riferimenti, ma il loro contenuto non è esaminabile, al di là di quanto sintetizzato in questo Syllabus relativamente a tali standard e libri.

Per ulteriori dettagli, fare riferimento al documento "Exam Structures and Rules" per CTAL - Agile Tester.

I criteri di ammissione per sostenere l'esame CTAL - Agile Tester sono:

- I candidati sono interessati al testing del software, in particolare in ambienti Agile.
- I candidati hanno ottenuto il certificato ISTQB® Foundation Level.

Tuttavia, è fortemente consigliato ai candidati anche di:

- Avere almeno un minimo di esperienza nello sviluppo software Agile o nel testing del software, come ad esempio sei mesi di esperienza come tester di sistema, user acceptance tester o membro di un team Agile.
- Frequentare un corso accreditato secondo gli standard ISTQB® (da uno dei Member Board riconosciuti da ISTQB).

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 ISTQB® Processes Management and Compliance.

0.8 Gestione degli Standard

Organizzazioni internazionali di standardizzazione come IEEE e ISO hanno pubblicato standard relativi alle caratteristiche di qualità e al testing del software. Tali standard sono citati in questo Syllabus. Lo scopo di questi riferimenti è fornire un quadro di riferimento (come nei riferimenti alla norma ISO 25010 relativamente alle caratteristiche di qualità) o fornire una fonte di informazioni aggiuntive, se il lettore lo desidera. Si noti che i Syllabi ISTQB® utilizzano i documenti degli standard come riferimenti. Il contenuto degli standard non è oggetto di esame. Per ulteriori informazioni sugli Standard, fare riferimento alla Sezione 7.

0.9 Livello di Dettaglio

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

- Obiettivi formativi generali che descrivono l'intento del Syllabus CTAL - Agile Tester

- Un elenco di termini (parole chiave) che i candidati devono essere in grado di ricordare
- Obiettivi di apprendimento per ogni area di conoscenza, che descrivono i risultati di apprendimento cognitivi che devono essere raggiunti
- Una descrizione dei concetti chiave, che include riferimenti a fonti riconosciute, come letteratura o standard accettati

Il contenuto del Syllabus non descrive l'intera area di conoscenza del testing del software; riflette il livello di dettaglio che deve essere coperto nei corsi di formazione per CTAL - Agile Tester. Si focalizza su approcci del test e tecniche di test che possono essere applicati a tutti i progetti software che seguono lo sviluppo software Agile.

Il Syllabus utilizza la terminologia (cioè denominazioni e significati) dei termini relativi al testing del software e alla Quality Assurance secondo il Glossario ISTQB®.

Per la terminologia delle discipline correlate, fare riferimento ai rispettivi glossari: IREB-CPRE (IREB-CPRE per il Requirements Engineering, [n.d.]), IEEE/Pascal (IEEE/Pascal per il software engineering, [n.d.]) e Agile Alliance per la terminologia Agile (Agile Alliance, [n.d.]).

0.10 Come è Organizzato questo Syllabus

Sono presenti sei capitoli con contenuti esaminabili. Nell'intestazione principale di ogni capitolo è specificato il tempo assegnato al capitolo. Per i corsi di formazione accreditati, il Syllabus richiede un minimo di 13 ore di istruzione, distribuite su almeno due giorni. Il tempo minimo di formazione è distribuito nei sei capitoli come segue:

- Capitolo 1: 60 minuti, Sfide relative alla Strategia di Test e all'Approccio del Test
- Capitolo 2: 60 minuti, Persone e Team
- Capitolo 3: 210 minuti, Test Management e Test Process Improvement
- Capitolo 4: 135 minuti, Shift-Left
- Capitolo 5: 285 minuti, Approcci e Tecniche di Test Agile
- Capitolo 6: 30 minuti, Test Automation e Strumenti di Test

1 Sfide relative alla Strategia di Test e all'Approccio del Test - 60 minuti

Parole Chiave

Acceptance Test-Driven Development, bug bash, edge case, testing end-to-end, testing formale, testing olistico, regression testing, tipo di test, Test-Driven Development

Parole Chiave Specifiche del Dominio

feature toggle, iterazione di hardening, iterazione, user journey

Obiettivi di Apprendimento:

1.1 Tipi di Test

CTAL-AT-1.1.1 (K2) Confrontare i tipi di test da eseguire durante e dopo un'iterazione

1.2 Testing End-to-End

CTAL-AT-1.2.1 (K2) Spiegare quando dovrebbe essere eseguito il testing end-to-end

1.3 Testing Formale e Testing Olistico

CTAL-AT-1.3.1 (K2) Confrontare i vantaggi e gli svantaggi del testing formale e del testing olistico

1.4 Approcci del Regression Testing

CTAL-AT-1.4.1 (K2) Distinguere tra i diversi approcci del regression testing

Introduzione all'Agile

Sebbene i fondamenti di base dell'Agile siano già descritti nel Syllabus del Foundation Level, questa introduzione approfondita integra i concetti essenziali non trattati in quel contesto ma fondamentali per comprendere le pratiche, i ruoli e la mentalità affrontati in questo Syllabus.

Lo sviluppo software Agile è diventato oggi uno degli approcci più diffusi nello sviluppo software. I suoi obiettivi principali sono adattarsi rapidamente ai cambiamenti e fornire costantemente valore ai clienti. Per raggiungere questi obiettivi, lo sviluppo software Agile adotta un approccio distinto dai modelli di sviluppo sequenziali, portando alla nascita del testing Agile. Il testing Agile va oltre il semplice testing del software; è un metodo collaborativo e integrato che incorpora la qualità in tutto il processo di sviluppo ed evidenzia la responsabilità dell'intero team nei confronti della qualità.

Sviluppo software Agile e Manifesto Agile. Nel 2001, un gruppo di persone che rappresentavano le metodologie di sviluppo software snelle più diffuse ha concordato una serie comune di valori e principi, che è diventata nota come Manifesto per lo sviluppo software Agile o Manifesto Agile (Beck; Beedle, et al., 2001). Il Manifesto Agile definisce i seguenti valori:

- Gli individui e le interazioni prima dei processi e degli strumenti. Lo sviluppo software Agile è fortemente incentrato sulle persone. Sono i team di persone a costruire il software, ed è attraverso la comunicazione e l'interazione continue, piuttosto che affidandosi a strumenti o processi, che i team possono lavorare in modo più efficace.
- Software funzionante piuttosto che documentazione esaustiva. Dal punto di vista del cliente, un software funzionante è molto più utile e prezioso di una documentazione eccessivamente dettagliata e offre l'opportunità di fornire un feedback rapido al team di sviluppo. Inoltre, poiché il software funzionante, anche con funzionalità ridotte, viene consegnato molto prima nel ciclo di vita dello sviluppo, lo sviluppo software Agile può offrire un significativo vantaggio in termini di time-to-market. Lo sviluppo software Agile è particolarmente utile in contesti aziendali in rapida evoluzione, dove i problemi o le soluzioni non sono chiari o dove l'azienda desidera innovare in nuovi domini problematici.
- Collaborazione con il cliente piuttosto che negoziazione contrattuale. I clienti spesso hanno difficoltà a specificare il sistema di cui hanno bisogno. La collaborazione diretta con il cliente aumenta la probabilità di comprendere esattamente ciò che il cliente richiede. Sebbene stipulare contratti con i clienti possa essere importante, lavorare in stretta e regolare collaborazione con loro è suscettibile di portare a un maggiore successo del progetto.
- Rispondere al cambiamento piuttosto che seguire un piano. Il cambiamento è inevitabile nei progetti software. L'ambiente in cui opera l'azienda, inclusi la legislazione, l'attività della concorrenza, i progressi tecnologici e altri fattori, può avere un impatto significativo sul progetto e sui suoi obiettivi. Questi fattori devono essere considerati nel processo di sviluppo. Pertanto, avere flessibilità nelle pratiche di lavoro per accogliere il cambiamento è più importante che attenersi rigidamente a un piano.

Principi Agile. I dodici principi del Manifesto Agile ne riassumono i valori fondamentali, ponendo l'accento sulla consegna di software di valore in modo tempestivo e frequente, sull'accettazione di requisiti che cambiano, su release frequenti e sulla collaborazione quotidiana tra gli stakeholder di business e gli sviluppatori. Essi danno priorità a individui motivati e supportati, alla comunicazione faccia a faccia, al software funzionante come misura chiave del progresso e allo sviluppo sostenibile a un ritmo costante. Sottolineano l'eccellenza tecnica, la semplicità, i team auto-organizzati e la riflessione regolare per migliorare continuamente l'efficacia e l'adattabilità.

Pratiche di sviluppo software Agile. I team Agile utilizzano varie pratiche prescrittive per mettere in atto questi valori e principi. Le pratiche comuni alla maggior parte delle organizzazioni Agile includono la creazione collaborativa di user story, le retrospettive, la continuous integration e la pianificazione di ogni iterazione, nonché della release complessiva.

Metodologie Agile. Diverse metodologie di sviluppo software Agile sono in uso nelle organizzazioni. Le più rappresentative sono:

- Scrum – un framework snello che organizza il lavoro in iterazioni di durata fissa chiamate sprint, ponendo l'accento su ruoli definiti, cerimonie e un product backlog con priorità (Sutherland et al., 2020)
- Extreme Programming (XP) – una metodologia incentrata su pratiche ingegneristiche quali Test-Driven Development, continuous integration e la stretta collaborazione, volte a migliorare la qualità e la reattività del software (Beck; Andres, 2004)
- Kanban – un metodo visivo di gestione del flusso di lavoro che limita il lavoro in corso, enfatizza la continuous delivery e utilizza lavagne per monitorare e ottimizzare il flusso (D. Anderson et al., 2010).

Definizione di testing Agile. Janet Gregory e Lisa Crispin, voci di spicco nella comunità Agile, definiscono il testing Agile come "pratiche di testing collaborativo che si svolgono dall'ideazione alla consegna, a supporto della consegna frequente di prodotti di qualità che aggiungono valore di business per i nostri clienti. Le attività di testing si focalizzano sulla prevenzione dei difetti piuttosto che sul loro rilevamento e lavorano per rafforzare e sostenere l'idea della responsabilità dell'intero team per la qualità" (Gregory; Crispin, 2019).

Questa definizione articola chiaramente i principi di base del testing Agile. Non si limita alle fasi finali di un progetto; al contrario, il testing è un'attività continua fin dall'inizio. Inoltre, il ruolo del testing è condiviso e si sposta dalla semplice identificazione dei difetti alla prevenzione proattiva dei difetti. Ciò incarna il whole team approach, in cui tutti coloro che sono coinvolti nel processo di sviluppo condividono la responsabilità della qualità.

Il testing Agile comprende un'ampia gamma di attività, tra cui – ma non solo – guidare lo sviluppo con esempi concreti, mettere in discussione idee e ipotesi con domande stimolanti, automatizzare i test per migliorare l'efficienza, eseguire testing esplorativo per individuare i difetti e valutare caratteristiche di qualità quali efficienza delle prestazioni, affidabilità e sicurezza. Queste attività dimostrano che nello sviluppo software Agile i ruoli del testing non si limitano all'esecuzione dei test; essi sono moderatori (facilitatori) della qualità e collaboratori.

Il Manifesto del Testing Agile. Il Manifesto del Testing Agile di Karen Greaves e Samantha Laing sintetizza i valori e i principi che guidano il testing Agile in cinque affermazioni concise. Questo manifesto evidenzia le differenze fondamentali rispetto al testing nello sviluppo software sequenziale e delinea la mentalità cruciale per i ruoli del testing in un contesto agile (Greaves et al., 2019):

- Testare durante tutto il processo piuttosto che alla fine. Nello sviluppo software sequenziale, il testing spesso inizia solo dopo che lo sviluppo è completato. Agile, invece, enfatizza il continuous testing durante tutto il ciclo di vita dello sviluppo, dall'analisi dei requisiti e dalla progettazione all'implementazione e al deployment. Questo approccio aiuta a identificare e correggere i difetti in anticipo, riducendo significativamente i costi e l'effort. Adottando l'approccio shift-left, l'intero team integra la qualità fin dall'inizio, rendendo il testing un processo di apprendimento continuo che rafforza la fiducia nel software in fase di test.

- Prevenire i difetti piuttosto che individuarli. L'obiettivo finale del testing Agile non è solo scoprire i difetti, ma impedire che si verifichino in primo luogo. Per raggiungere questo obiettivo, i ruoli del testing partecipano attivamente alle discussioni iniziali, come il chiarimento dei requisiti e la revisione dei progetti, identificando e mitigando così in modo proattivo i potenziali problemi.
- Verificare la comprensione piuttosto che controllare la funzionalità. Oltre alla semplice verifica che il software svolga le funzioni dichiarate, è fondamentale testare la comprensione di fondo dello scopo del software e delle esigenze degli utenti. Ciò comporta la valutazione del sistema dal punto di vista dell'utente, spesso attraverso user story e scenari, per garantire che sia in linea con il valore di business.
- Costruire il sistema migliore piuttosto che metterlo in difficoltà. La funzione primaria di un ruolo del testing non è quella di sfruttare le debolezze del sistema o di metterlo intenzionalmente in difficoltà. Al contrario, i ruoli del testing collaborano con gli sviluppatori per aiutare a costruire un sistema della massima qualità. Ciò include fornire feedback costruttivi, proporre progetti testabili e condividere idee per il miglioramento continuo della qualità.
- Responsabilità del team per la qualità piuttosto che responsabilità del tester. La qualità non è responsabilità esclusiva di un singolo individuo o di un team di test dedicato. Si tratta invece di un impegno condiviso tra tutti coloro che sono coinvolti nel processo di sviluppo, inclusi sviluppatori, rappresentanti del business, tester e altri stakeholder. Il testing Agile promuove questa cultura di responsabilità condivisa, incoraggiando ogni membro del team a partecipare attivamente alle attività relative alla qualità.

L'importanza e l'impatto del testing Agile. Questi principi fondamentali del testing Agile sono essenziali per garantire la qualità del software nello sviluppo software Agile, che dà priorità alla reattività e al feedback rapido. Nello sviluppo sequenziale del software, il testing diventa spesso un collo di bottiglia del progetto o un quality gate finale appena prima della release. Tuttavia, in Agile, il testing è perfettamente integrato come parte fondamentale e indispensabile del flusso di lavoro dello sviluppo. Questa integrazione consente ai team di fornire costantemente software di alta qualità.

Il testing Agile rafforza inoltre in modo significativo la collaborazione e la comunicazione all'interno del team. I ruoli del testing lavorano fianco a fianco con gli sviluppatori e gli stakeholder di business per chiarire i requisiti, identificare precocemente potenziali difetti e incoraggiare tutti a condividere un obiettivo comune di qualità. Questo ambiente collaborativo non solo contribuisce alla prevenzione dei difetti, ma svolge anche un ruolo vitale nell'aumentare la produttività del team e l'efficienza complessiva. Il ruolo del testing non è necessariamente legato a una posizione specifica all'interno dell'organizzazione, ma può essere svolto da chiunque in qualsiasi momento.

In sostanza, il testing Agile è più di una semplice raccolta di tecniche di test; rappresenta un profondo cambiamento nella cultura e nella mentalità dello sviluppo software. È essenziale focalizzarsi sul valore per il cliente, adattarsi in modo flessibile al cambiamento e fornire continuamente prodotti di alta qualità sotto la responsabilità collettiva dell'intero team.

1.1 Tipi di Test

I tipi di test vengono selezionati in base agli obiettivi del test e al contesto specifici dell'iterazione, inclusi i criteri di accettazione delle user story, i rischi di qualità e le esigenze di feedback. Tutti i tipi di test possono essere utilizzati in qualsiasi fase del progetto, ma il loro utilizzo può variare durante e dopo l'iterazione. Qualsiasi decisione presa dovrebbe riflettersi nella Definition of Done e nell'approccio del test.

Durante l'iterazione, il testing funzionale verifica che le funzionalità implementate soddisfino i relativi requisiti funzionali e i criteri di accettazione. Dopo l'iterazione, il testing funzionale esteso può includere il testing funzionale dell'integrazione di diverse funzionalità, comprese quelle sviluppate da altri team.

Durante l'iterazione, il testing non-funzionale è tipicamente limitato al testing dell'efficienza delle prestazioni, dell'usabilità o della sicurezza di base delle nuove funzionalità. Dopo l'iterazione, è possibile eseguire un testing non-funzionale completo su ambienti integrati o di pre-produzione (staging). Ulteriore testing esplorativo e di usabilità con un gruppo più ampio possono essere eseguiti anche dopo l'iterazione per validare l'idoneità del prodotto allo scopo. Durante l'iterazione, il testing white-box viene utilizzato prima o durante la codifica, ad esempio come parte del Test-Driven Development (TDD). Ciò supporta il rilevamento tempestivo dei difetti e aiuta a mantenere un codice di base stabile. Raramente viene eseguito come attività separata post-iterazione; tuttavia, la copertura del codice può essere misurata durante il testing black-box per fornire ulteriori informazioni sull'efficacia e la completezza dei test black-box.

Durante l'iterazione, il testing black-box si focalizza sulla verifica di funzionalità nuove o modificate rispetto alle specifiche definite tipicamente come criteri di accettazione delle user story. Ciò è coerente con il testing funzionale che utilizza l'Acceptance Test-Driven Development (ATDD), contribuendo a garantire che la funzionalità consegnata soddisfi le aspettative degli stakeholder. Dopo l'iterazione, può essere eseguito il testing black-box più ampio a livello di sistema, inclusi regression testing e testing end-to-end.

Durante l'iterazione, il testing esplorativo svolge un ruolo fondamentale. Questo approccio specifico al testing funzionale integra sia il testing white-box che il testing black-box, individuando rischi sconosciuti e comportamenti emergenti attraverso sessioni strutturate e time-boxed.

Per informazioni sul collegamento tra i tipi di test e i quadranti del testing, consultare la *Sezione 3.1.2*.

1.2 Testing End-to-End

Nello sviluppo software Agile, il testing end-to-end viene eseguito quando aggiunge valore verificando l'integrazione tra sistemi e flusso di lavoro dell'utente. Questo include il testing dell'intera applicazione e, se rilevante, di più sistemi o servizi. Tuttavia, come misura di mitigazione, dovrebbe essere bilanciato rispetto al rischio associato alla catena E2E e alla specifica user story, poiché è costoso da mantenere, richiede tempo per essere eseguito e fornisce un feedback diagnostico limitato. Il testing end-to-end viene tipicamente eseguito in prossimità di una release, durante iterazioni di hardening, oppure quando vengono introdotti punti di integrazione a livello di sistema nuovi o ad alto rischio di failure. Sebbene le iterazioni di hardening siano utilizzate in alcune organizzazioni, possono indicare un'attenzione insufficiente alla qualità durante le iterazioni regolari. Il testing end-to-end è particolarmente utile nei sistemi distribuiti, negli user journey complessi o negli scenari con dipendenze di terze parti che non possono essere testate in modo affidabile in isolamento. Il suo scopo è validare i flussi utente critici in un ambiente di pre-produzione, non rilevare difetti di basso livello.

Diversi fattori influenzano la decisione di includere test end-to-end:

- **Rischio.** L'impatto del rischio e la probabilità del rischio di un failure a livello di sistema giustificano l'esecuzione di testing end-to-end mirato, anche se la fiducia è stata acquisita tramite test di livello inferiore.
- **Tempo di feedback.** Affidarsi eccessivamente ai test end-to-end rallenta la consegna. I team spesso preferiscono test più rapidi a livelli di test inferiori e selezionano solo alcuni flussi chiave per la copertura end-to-end.

- Maturità della pipeline di continuous integration/continuous delivery (CI/CD). Quando i team dispongono di buoni test automation, osservabilità e monitoraggio in produzione, la fiducia può spostarsi verso il feedback dalla produzione, i feature toggle o il testing A/B come alternative, o complemento, al testing end-to-end esteso.

Dal punto di vista della strategia di test, la piramide di test rimane valida: feedback ampio e veloce alla base (testing di componente), un numero minore di test a livello di servizio e test end-to-end minimi al vertice. Deviare da questa struttura può aumentare i costi e i rischi senza un valore proporzionato.

Nei microservizi e nelle architetture distribuite, il testing di contratti offre un'alternativa al testing end-to-end esteso. Il testing di contratti verifica che i servizi comunichino correttamente tramite la validazione delle interfacce concordate (contratti) tra consumer e provider di servizi. A differenza dei test end-to-end che richiedono l'esecuzione di tutti i servizi, i test di contratto possono essere eseguiti in modo indipendente, fornendo un feedback più rapido e una diagnosi dei failure più semplice. Il testing di contratti consumer-driven, in cui i consumer definiscono le proprie aspettative e i provider verificano di soddisfarle, è particolarmente efficace in ambienti Agile in cui i servizi evolvono in modo indipendente. Il testing di contratti integra, anziché sostituire, un piccolo insieme di test end-to-end che validano gli user journey critici.

Il testing di integrazione dovrebbe essere preferibilmente eseguito a livello di interfaccia, a meno che non siano esplicitamente impattati comportamenti a livello di sistema. Il testing end-to-end dovrebbe essere allineato agli obiettivi di qualità definiti dagli stakeholder e non utilizzato in modo indiscriminato (ISO 29119-6 Software and systems engineering – Software testing, 2021).

I test end-to-end scriptati possono non rilevare comportamenti inattesi derivanti dall'utilizzo reale. Il testing olistico (vedere la *Sezione 1.3*) e il testing esplorativo (vedere la *Sezione 5.1.5*) integrano questi test consentendo ai tester di esaminare la qualità da più prospettive e utilizzare il testing basato sull'esperienza per contribuire a individuare comportamenti inattesi, difetti di usabilità ed edge case che gli script predefiniti potrebbero non rilevare.

1.3 Testing Formale e Testing Olistico

Il **testing formale** enfatizza precisione e formalizzazione. Prevede la derivazione dei test case da requisiti esplicitamente definiti. Viene tipicamente utilizzato in contesti che richiedono rigorosa conformità o controllo del rischio (ad esempio, regression testing, quando la regressione è considerata un rischio).

I vantaggi del testing formale includono:

- Tracciabilità: la tracciabilità è garantita tra requisiti e test case.
- Allineamento: i test case sono allineati con le metriche di copertura.
- Verifica: il testing formale aiuta nella verifica, testando se il sistema è stato costruito correttamente.
- Conformità: i test case possono essere sottoposti a ispezione per garantire la conformità a standard e linee guida.
- Ripetibilità: poiché i test case sono documentati formalmente, la ripetibilità è garantita.
- Test Automation: formalizzazione e precisione facilitano la transizione alla test automation.

Le sfide del testing formale includono:

- Eccessiva rigidità: tende a limitare l'esplorazione e la reattività al cambiamento.

- Eccessiva dipendenza dai requisiti documentati: la base di test è tipicamente incompleta, incoerente o errata. Quando i test case vengono creati esclusivamente a partire da requisiti espliciti scadenti o difettosi, ciò comporta una copertura incompleta, test case non allineati e una maggiore probabilità di difetti non rilevati.
- Mancata individuazione dell'imprevisto: focalizzandosi solo sui risultati attesi, si possono trascurare rischi e failure emergenti.
- Feedback ritardato: spesso viene eseguito in una fase avanzata del ciclo di test, il che ritarda l'individuazione di problemi critici.
- Basso coinvolgimento cognitivo: incoraggia l'esecuzione meccanica dei test invece di un'analisi ponderata.

Il **testing olistico** (Gregory, 2021) va oltre le attività di test pianificate per abbracciare una visione della qualità a livello di sistema. Incorpora il continuous testing durante tutto il ciclo di vita dello sviluppo software e include un'ampia gamma di prospettive: cliente, team e organizzazione. Si focalizza su collaborazione, consapevolezza del contesto, pensiero sistemico (systems thinking) (vedere (ISTQB-ATLAS, v1.0), sezione 3.2.1) ed esplorazione.

I vantaggi del testing olistico includono:

- Affrontare diverse prospettive: combinare contemporaneamente punti di vista basati su processo, prodotto e utente.
- First Time Right (FTR): garantire che la qualità sia integrata, non solo testata a posteriori.
- Continuous Testing: combina verifica (costruire correttamente il sistema) e validazione (costruire il sistema corretto).

Le sfide del testing olistico includono:

- Mancanza di focalizzazione: senza una struttura disciplinata, i team potrebbero eseguire test su ampia scala ma trascurare rischi critici o regressioni.
- Difficoltà di misurazione: la valutazione della qualità è sensibile al contesto e manca di metriche fisse, mettendo a dura prova la fiducia degli stakeholder.
- Richiede un cambiamento culturale: si basa su una forte collaborazione e sul pensiero sistemico e, senza una leadership di supporto e la maturità del team, l'adozione potrebbe fallire.
- Ambiguità nelle responsabilità: la titolarità da parte dell'"intero team" può indebolire la responsabilità se non gestita con attenzione.
- Forte dipendenza dal contesto: non è universalmente applicabile e potrebbe non soddisfare i vincoli normativi o contrattuali che richiedono una verifica più formale.
- Problemi di Test Automation: la mancanza di formalità mette a rischio una corretta test automation.

Per massimizzare i benefici di questi due approcci, essi possono essere utilizzati in modo complementare. Il testing formale dovrebbe focalizzarsi su componenti critici, aree ad alto rischio o requisiti normativi e legali, ed essere automatizzato ove possibile per minimizzare l'effort ripetitivo. Il testing olistico dovrebbe essere utilizzato per il Quality Control generale, il feedback precoce e l'esplorazione di scenari complessi, dando priorità alle aree con il maggiore impatto sugli utenti.

1.4 Approcci del Regression Testing

Nello sviluppo software Agile, il regression testing deve essere organizzato in modo da supportare la continuous delivery e il cambiamento rapido. Diversi approcci al regression testing possono essere utilizzati singolarmente o in combinazione, a seconda del contesto, delle capacità del team, del profilo di rischio e della maturità della test automation:

- Regression testing incrementale: si basa su una test suite automatizzata che viene mantenuta ed eseguita continuamente. Invece di rieseguire una regression test suite completa solo alla fine di un'iterazione o di una release, un sottoinsieme di regression test viene attivato automaticamente a ogni commit di codice o evento di integrazione, selezionato in base all'impatto del rischio, alla probabilità del rischio, alla posizione e alle dipendenze della modifica del codice. Ciò consente il rilevamento tempestivo dei difetti di regressione e supporta cicli di feedback continui.
- Regression testing basato sul rischio: utilizza frequenti workshop di valutazione del rischio o discussioni del team per identificare le aree in cui è probabile che le modifiche causino difetti, e guida sia l'effort di regressione automatizzato sia quello manuale. Questo metodo aiuta a decidere dove la test automation non è necessaria e dove l'indagine umana aggiunge maggiore valore.
- Regression testing orientato a DevOps: integra il regression testing nella pipeline di CI/CD, con smoke test che fungono da quality gate e regression test ad alta priorità eseguiti automaticamente dopo la distribuzione in ambienti di pre-produzione o persino di produzione; feature toggle e canary release consentono un'esposizione selettiva, mentre osservabilità e monitoraggio possono, in alcuni casi, sostituire il regression testing tradizionale.
- Regression testing esplorativo: viene applicato in modo continuo o a intervalli regolari, integrando la test automation mediante le intuizioni dei tester per esplorare le interazioni tra funzionalità e verificare la presenza di difetti inattesi.
- Regression testing collaborativo: include attività di test come bug bash o walkthrough da parte dell'intero team su flussi critici, aumentando la consapevolezza del team sullo stato attuale del prodotto e sulle aree di rischio. Può essere time-boxed e utilizzato quando la test automation non è ancora matura, oppure come esercizio di apprendimento e identificazione del rischio.

La tracciabilità tra user story, requisiti e criteri di accettazione e i relativi test case svolge un ruolo chiave nel regression testing. Consente ai team di identificare rapidamente quali test devono essere aggiornati o rieseguiti quando vengono apportate modifiche, riducendo così l'effort superfluo e focalizzando il regression testing sulle funzionalità impattate. Evidenzia inoltre le lacune della copertura, aiutando i team a bilanciare testing automatizzato, testing esplorativo e regression testing collaborativo. La tracciabilità leggera (ad esempio, l'associazione dei test alle user story in uno strumento di backlog) consente ai team di mantenere la fiducia nelle release frequenti senza rallentare la consegna.

2 Persone e Team - 60 minuti

Parole Chiave

Acceptance Test-Driven Development, Behavior-Driven Development, tissue tester

Obiettivi di Apprendimento:

2.1 Whole Team Approach

CTAL-AT-2.1.1 (K2) Confrontare generalizzazione e specializzazione all'interno di un team

CTAL-AT-2.1.2 (K2) Fornire esempi di come motivare i rappresentanti del business a svolgere attività di test

CTAL-AT-2.1.3 (K2) Riassumere in che modo il whole team approach possa aiutare il team di sviluppo

2.2 Tissue Tester

CTAL-AT-2.2.1 (K2) Spiegare come e quando utilizzare i tissue tester

Introduzione

Nei progetti di sviluppo software Agile, le persone e gli strumenti costituiscono la base per fornire qualità in tempi rapidi. Il whole team approach sottolinea che la qualità è una responsabilità condivisa tra sviluppatori, tester e rappresentanti del business che collaborano strettamente. In genere, i team sono composti da membri multidisciplinari o interfunzionali. Mentre gli specialisti portano una competenza solida, i generalisti si adattano a diversi ruoli, consentendo una maggiore flessibilità e una condivisione del carico di lavoro. Coinvolgere e motivare i rappresentanti del business a contribuire al testing attraverso un coinvolgimento precoce nella definizione dei criteri di accettazione, nella revisione dei risultati dei test e nella partecipazione al testing esplorativo, migliora la qualità del prodotto e l'allineamento con gli obiettivi di business. Questa collaborazione avvantaggia anche gli sviluppatori, poiché la condivisione delle informazioni sul testing riduce i difetti e accelera il feedback. I team agili possono anche coinvolgere tester esterni: tester esterni, che forniscono una prospettiva "nuova" (fresh eyes) per individuare prima i difetti. Sapere come e quando utilizzare tali tester aiuta a fornire un feedback equilibrato e imparziale senza rallentare la consegna.

2.1 Whole Team Approach

2.1.1 Generalizzazione e specializzazione all'interno di un team

L'essenza del whole team approach risiede nel fatto che tester, sviluppatori e rappresentanti del business lavorano insieme come un team Agile in ogni fase del processo di sviluppo. La generalizzazione e la specializzazione dei ruoli all'interno di un team Agile influiscono direttamente su come le attività di test vengono distribuite, gestite e migliorate. Agile promuove un whole team approach alla qualità, in cui tutti sono responsabili della qualità indipendentemente dal titolo o dalla posizione.

I tester Agile sono generalmente specialisti con "competenze a T", ovvero con una preparazione trasversale. Possiedono una solida competenza nel testing e una vasta conoscenza in diverse discipline come la programmazione, la business analysis e il DevOps. Ciò favorisce la collaborazione interfunzionale e consente flessibilità nella composizione del team.

La generalizzazione consente a più membri del team di svolgere una determinata attività. Ciò supporta un ritmo sostenibile, riduce i colli di bottiglia e migliora la resilienza del team. Incoraggia inoltre la responsabilità condivisa della qualità, che è essenziale negli ambienti di continuous delivery. La specializzazione dei ruoli esiste ancora, in particolare per le attività di test che richiedono competenze specifiche, come le tecniche di test, il testing esplorativo, la test automation o il testing di usabilità. Tuttavia, i team Agile si sforzano di ridurre al minimo i silos, in cui gli individui lavorano entro confini di ruolo ristretti, incoraggiando la condivisione delle conoscenze e il lavoro in coppia.

Il mob programming e il mob testing (vedere la *Sezione 5.2.1*) rendono meno netti i ruoli all'interno del team. Questi approcci riuniscono l'intero team, inclusi i rappresentanti del business, gli sviluppatori e i tester, per lavorare su un unico compito contemporaneamente, piuttosto che dividere il lavoro rigorosamente in base a ruoli specializzati.

I tester spesso assumono un ruolo guida nell'aiutare gli altri ad applicare pratiche incentrate sulla qualità, come il BDD, il testing esplorativo e la test automation. Allo stesso tempo, possono imparare dagli sviluppatori i dettagli di implementazione per valutare meglio i rischi e progettare i test. In questo modo, i tester preservano la loro specializzazione (conoscenza approfondita del testing) sviluppando al contempo competenze più ampie che contribuiscono alla generalizzazione all'interno del team. L'equilibrio tra specializzazione e generalizzazione dovrebbe essere determinato dal contesto. Domini complessi o

ambienti regolamentati possono richiedere ruoli più specializzati, mentre le startup snelle possono trarre vantaggio da generalisti con competenze ampie. Indipendentemente dalla struttura, sono essenziali una comunicazione chiara e il rispetto reciproco tra i membri del team. I tester dovrebbero contribuire alle discussioni fin dall'inizio del progetto per garantire che la qualità sia integrata già dall'inizio, anziché essere valutata solo in un secondo momento.

Negli ambienti Agile con più team (ad esempio, SAFe, LeSS), il whole team approach va oltre i singoli team. Il coordinamento tra i team sugli standard di qualità, il testing di integrazione e la responsabilità dei difetti ai confini dei team diventa essenziale. I tester partecipano spesso a “community of practice”, in cui vengono condivise le conoscenze tra i team, vedi (ISTQB-ATLAS, v1.0).

2.1.2 Motivare i rappresentanti del business a partecipare al testing

La qualità è una responsabilità condivisa da tutto il team. I rappresentanti del business definiscono la qualità del prodotto, ma dovrebbero anche partecipare alle attività di test. Per motivarli a testare, occorre coinvolgerli sin dalle prime fasi del processo di test, in particolare nella definizione dei criteri di accettazione e nella formulazione di test orientati al business. Il loro coinvolgimento in queste discussioni iniziali garantisce che ciò che viene consegnato sia in linea con il valore per il cliente e con le intenzioni degli stakeholder.

Il testing non dovrebbe essere presentato come una fase separata, ma piuttosto come un'attività collaborativa e continua. Pratiche come l'Acceptance Test-Driven Development, il Behavior-Driven Development e la specifica per esempi aiutano a consolidare il ruolo del rappresentante del business nel chiarire il comportamento attraverso esempi concreti e testabili. Queste pratiche rendono il testing visibile e rilevante per le preoccupazioni del business, invitando i rappresentanti del business a un dialogo significativo con i membri del team e gli altri stakeholder.

Incorporare la testabilità nelle user story durante l'affinamento (il perfezionamento progressivo) del backlog garantisce che i criteri di accettazione siano concreti piuttosto che astratti, legati a risultati di test reali e verificabili. Lavorare a stretto contatto con tester e sviluppatori in queste sessioni permette ai rappresentanti del business di vedere come requisiti poco chiari portino ad ambiguità e rework (rilavorazioni dei prodotti di lavoro), e come esempi ben formulati portino a una consegna più veloce e prevedibile.

La visibilità è un fattore chiave di motivazione. I dashboard che mostrano la copertura, le tendenze dei difetti e le metriche relative ai rischi di qualità rendono più facile per i rappresentanti del business comprendere l'attuale qualità del prodotto e dove il testing aggiunge valore. Le metriche di business, come la soddisfazione del cliente, i costi di assistenza o l'adozione delle funzionalità, aiutano i rappresentanti del business a considerare il testing come parte della validazione del prodotto, non solo della verifica. Queste informazioni aiutano a collegare il testing direttamente alle decisioni sul prodotto e alla sua idoneità al rilascio.

La responsabilità condivisa viene rafforzata nella collaborazione quotidiana. Durante la pianificazione e il perfezionamento delle iterazioni, i rappresentanti del business dovrebbero contribuire attivamente a definire cosa debba significare “done” per una particolare user story, inclusi la copertura e i criteri di accettazione. La loro presenza durante attività di test come i bug bash o il testing di usabilità garantisce che i cicli di feedback rimangano brevi e basati sul valore di business, aumentando al contempo la loro motivazione mentre assistono ai frutti del loro lavoro.

Infine, riconoscere il contributo dei rappresentanti del business al testing durante le retrospettive e le demo rafforza il loro ruolo nella qualità e li mantiene coinvolti. Con aspettative chiare, accesso alle informazioni rilevanti e continue opportunità di influenzare i risultati, i rappresentanti del business sono più

propensi a considerare il testing come una parte fondamentale delle loro responsabilità e una potente leva per fornire prodotti di successo.

2.1.3 Supportare gli sviluppatori

Il whole team approach aiuta gli sviluppatori integrando una collaborazione focalizzata sulla qualità in tutto il ciclo di vita dello sviluppo software. L'interazione continua tra sviluppatori, tester e rappresentanti del business crea una comprensione condivisa degli obiettivi di qualità e dei potenziali rischi, portando a decisioni tecniche e di business migliori. Gli sviluppatori traggono vantaggio da feedback più precoci sui test, che riducono i costi dei difetti e supportano azioni correttive rapide, in particolare quando vengono applicate pratiche come l'Acceptance Test-Driven Development e il Behavior-Driven Development.

I tester, lavorando a stretto contatto con gli sviluppatori, aiutano a chiarire i criteri di accettazione attraverso esempi, portando a specifiche per esempi migliorate e a un migliore allineamento tra il codice e le aspettative del business.

I tester possono guidare gli sviluppatori nella scrittura di test di componente e test di integrazione dei componenti più efficaci, al fine di raggiungere livelli più elevati di copertura del codice. Man mano che il testing diventa un'attività continua anziché un processo di fase avanzata, gli sviluppatori acquisiscono la sicurezza necessaria per rifattorizzare e far evolvere il codice in modo sicuro. Automatizzare i test nelle prime fasi del ciclo di vita dello sviluppo software consente un rilevamento più rapido dei difetti di integrazione e facilita l'implementazione delle pipeline di CI/CD.

Inoltre, la mentalità del team nel suo insieme abbate i silos di ruolo, creando un ambiente in cui i tester contribuiscono alle review del codice dal punto di vista della qualità e del rischio, e gli sviluppatori partecipano a sessioni di testing esplorativo. Questo ambiente di apprendimento reciproco non solo eleva la competenza complessiva del team in materia di test, ma promuove anche una responsabilità collettiva per la qualità del prodotto. Gli sviluppatori acquisiscono inoltre informazioni sul comportamento degli utenti reali partecipando a testing esplorativo, sessioni di test di usabilità o analizzando i dati analitici di produzione, il che li aiuta a costruire software più in linea con le esigenze degli utenti.

Un altro modo in cui il whole team approach supporta gli sviluppatori è il coinvolgimento precoce di Operations, degli specialisti della sicurezza e dei tester per affrontare le considerazioni relative al deployment, al monitoraggio e alla conformità durante lo sviluppo. Questa collaborazione aiuta gli sviluppatori a progettare soluzioni più facili da distribuire, monitorare e proteggere in produzione, riducendo i rework in fase avanzata e i problemi operativi. Incorporando queste prospettive fin dall'inizio, gli sviluppatori possono fare scelte architetturali che bilanciano la funzionalità con le caratteristiche di qualità. Questo, in definitiva, accelererà la consegna aumentando al contempo la fiducia nella prontezza del sistema per l'uso nel mondo reale.

2.2 Tissue Tester

I tissue tester (noti anche come tester "fresh eyes") sono tester temporanei, esterni al team, che forniscono feedback rapido e informale su un prodotto o una funzionalità in fase di sviluppo, tipicamente utilizzati in settori volatili come l'industria dei videogiochi. Sono più efficaci se utilizzati nelle prime fasi della progettazione o dell'implementazione delle funzionalità, prima che i team si impegnino in decisioni di sviluppo o infrastrutturali costose. L'idea è esporre brevemente una nuova idea, interfaccia o interazione a qualcuno che non la conosce, per individuare difetti di usabilità, comportamenti confusi o flussi fuorvianti.

I tissue tester sono in genere persone esterne al team o al progetto (ad esempio, un collega di un altro team) e dovrebbero essere utilizzati con la consapevolezza che il loro feedback può variare

significativamente da individuo a individuo e che il loro coinvolgimento è di breve durata. Dopo aver visto il prodotto una volta, non sono più realmente imparziali, quindi il loro valore speciale come "fresh eyes" diminuisce (vedere la *Sezione 4.1.4*). Questa metafora sottolinea la natura leggera e transitoria della pratica. Un particolare tissue tester non dovrebbe essere utilizzato nuovamente dopo aver visto il progetto in fase di test, perché le sue reazioni saranno influenzate dalla conoscenza pregressa. verrebbe meno l'essenziale prospettiva di chi osserva il prodotto per la prima volta.

I tissue tester sono più adatti per il testing esplorativo, il testing di usabilità o la verifica delle ipotesi, dove un feedback rapido è prezioso ma è troppo presto per una validazione rigorosa. Ad esempio, prima di perfezionare un'interazione o un flusso di schermate, uno sviluppatore o un tester potrebbe guidare un tissue tester attraverso di esso per identificare potenziali problemi. È importante non spiegare troppo e quindi non introdurre bias, poiché ciò distruggerebbe la loro prospettiva di chi osserva il prodotto per la prima volta.

Il tissue testing non dovrebbe sostituire il testing strutturato. Piuttosto, dovrebbe precederlo o integrarlo. Il suo valore principale sta nel rivelare tempestivamente eventuali discrepanze tra l'intento progettuale e l'effettiva comprensione da parte dell'utente. Poiché è rapido da eseguire e non richiede il reclutamento di utenti esterni, il tissue testing è utile nello sviluppo software Agile e nei contesti DevOps.

3 Test Management e Test Process Improvement - 210 minuti

Parole Chiave

sviluppo software Agile, chaos testing, controllo dei test, monitoraggio dei test, pianificazione dei test, test process improvement, reporting del test, strategia di test, quadranti del testing

Parole Chiave Specifiche del Dominio

canary release, community of practice, dark launch, Definition of Done, feature toggle, Goal-Question-Metric, metrica

Obiettivi di Apprendimento:

3.1 Pianificazione dei Test

CTAL-AT-3.1.1 (K2) Riassumere come eseguire la pianificazione dei test nello sviluppo software Agile

CTAL-AT-3.1.2 (K4) Delineare una strategia di test del progetto utilizzando i quadranti del testing

3.2 Monitoraggio dei Test e Controllo dei Test

CTAL-AT-3.2.1 (K2) Spiegare come eseguire il monitoraggio dei test e il controllo dei test nello sviluppo software Agile

3.3 Reporting del Test

CTAL-AT-3.3.1 (K2) Confrontare i diversi tipi di copertura che possono essere utilizzati per il reporting del test nello sviluppo software Agile

3.4 Test Process Improvement

CTAL-AT-3.4.1 (K4) Selezionare misure appropriate di Test Process Improvement basate su metriche nello sviluppo software Agile

CTAL-AT-3.4.2 (K2) Spiegare come eseguire il Test Process Improvement nello sviluppo software Agile

Introduzione

Nei progetti di sviluppo software Agile, le attività di test sono integrate in cicli di test brevi e iterativi, che richiedono una pianificazione dei test adattiva che si evolva in base alle priorità del prodotto e del team. La strategia di test di un progetto è supportata dai quadranti del testing (Gregory; Crispin, 2019), che vengono utilizzati per selezionare tipi di test, livelli di test, approcci del test e tecniche di test appropriati per ottenere una copertura equilibrata delle esigenze orientate al business e orientate alla tecnologia.

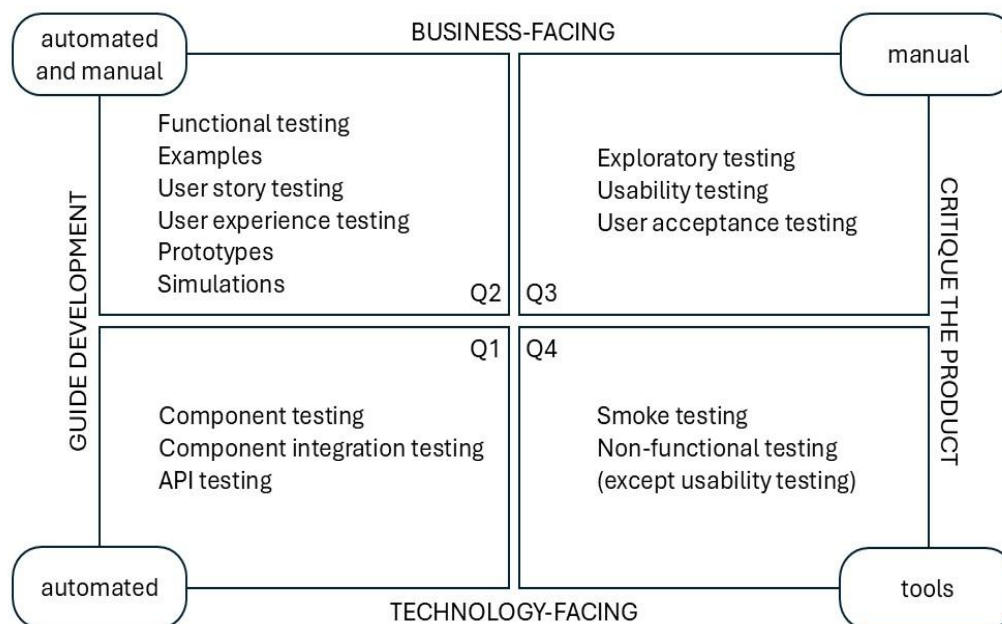


Fig. 1. Modello dei quadranti del testing

Il quadrante 1 contiene test orientati alla tecnologia che aiutano a guidare lo sviluppo. Essi verificano piccoli frammenti di codice. Il quadrante 2 contiene test orientati al business che aiutano a guidare lo sviluppo. Essi verificano che il prodotto si comporti come previsto dal business. Il quadrante 3 contiene test orientati al business che valutano criticamente il prodotto. Essi verificano quanto bene il prodotto funzioni per gli utenti. Il quadrante 4 contiene test orientati alla tecnologia che valutano criticamente il prodotto. Essi verificano le caratteristiche di qualità del prodotto.

I principi Agile definiscono ulteriormente la strategia di test promuovendo una pianificazione adattiva, un feedback tempestivo e una continua prioritizzazione basata sul rischio, al fine di allineare il testing al valore di business. Il monitoraggio dei test e il controllo dei test sono continui e utilizzano metriche leggere e in tempo reale per tracciare i progressi, individuare i rischi e adeguare le priorità. Il reporting del test fornisce metriche di copertura significative e contestualizzate, quali la copertura dei requisiti, del codice e dei rischi, per offrire agli stakeholder informazioni chiare e inequivocabili sullo stato della qualità. Il test process improvement continuo è implicito in Agile, che utilizza le metriche per identificare i colli di bottiglia, perfezionare le pratiche e migliorare l'efficienza. Avviando miglioramenti tempestivi e frequenti, i team possono rispondere al cambiamento, mantenere la qualità e massimizzare il valore del testing all'interno di ogni iterazione.

3.1 Pianificazione dei Test

3.1.1 Pianificazione dei test nello sviluppo software Agile

In Agile, la pianificazione dei test avviene tipicamente a due livelli principali: pianificazione dell'iterazione e pianificazione della release. La strategia di test del progetto influisce su queste attività di pianificazione.

Pianificazione dell'iterazione

Durante la pianificazione dell'iterazione, il team revisiona il product backlog e seleziona le user story per l'iterazione.

I tester possono contribuire facilitando sessioni di risk-storming con i rappresentanti del business e gli sviluppatori per identificare i rischi e prioritizzare le attività di test, definire le condizioni di test di alto livello e perfezionare i criteri di accettazione. La pianificazione dell'iterazione include la selezione di tipi di test, livelli di test, approcci del test e tecniche di test appropriati per la data iterazione. I tester utilizzano le loro conoscenze ed esperienze per aiutare il team a fare le scelte giuste. I quadranti del testing possono essere utilizzati per supportare questo compito.

La pianificazione dell'iterazione deve tenere conto dell'ambiente di test e dei dati di test necessari, per garantire che tutto sia pronto all'inizio dell'iterazione. Poiché il testing non costituisce una fase separata nell'Agile e può avvenire in qualsiasi momento durante l'iterazione, gli ambienti di test e i dati di test dovrebbero essere stabili e disponibili fin dall'inizio dell'iterazione. Il test plan include tipicamente pratiche di continuous integration e continuous testing. Il testing manuale integra la test automation occupandosi delle aree che richiedono il giudizio umano, come il testing di usabilità e il testing esplorativo.

La pianificazione dell'iterazione include l'accordo o la revisione della Definition of Done a livello di team, specificando i criteri di uscita per le user story e altri elementi del backlog all'interno dell'iterazione.

Chiarisce inoltre in che modo l'esecuzione dei test, i risultati dei test e le informazioni sulla copertura contribuiscono al raggiungimento della Definition of Done per ogni elemento e per il risultato dell'iterazione.

Pianificazione della release

Durante la pianificazione della release, la pianificazione dei test opera a un livello più ampio rispetto alla pianificazione dell'iterazione. Il team identifica gli obiettivi di business di alto livello e valuta il product backlog per determinare l'ambito e la sequenza delle funzionalità da consegnare. Dal punto di vista del testing, ciò comporta la valutazione dei rischi di qualità nell'ambito della release, la definizione degli obiettivi del test a livello di release e la determinazione dei livelli di test, dei tipi di test, degli approcci del test e delle tecniche di test appropriati necessari per mitigare tali rischi.

I tester contribuiscono identificando le dipendenze tra le funzionalità, i vincoli sugli ambienti di test e sui dati, e i punti di integrazione che richiedono una validazione specifica. Aiutano a stimare l'effort di test complessivo utilizzando tecniche basate su metriche e su esperti, che supportano la capacità del team di pianificare la capacità su più iterazioni.

Il team delinea una strategia di test a livello di release che copra almeno sia il testing funzionale che quello non-funzionale, come questi saranno incorporati nelle iterazioni e come sarà gestito il regression testing. Assicura una pianificazione precoce per il testing basato sull'esperienza e identifica dove è necessario coinvolgere stakeholder esterni, come clienti o esperti di conformità. Il tester, che spesso agisce come moderatore, può guidare il team attraverso esercizi collaborativi (ad esempio, workshop di mappatura mentale o sessioni di risk-storming) per progettare la strategia.

La pianificazione della release include un accordo sulla Definition of Done a livello di release, compresi i criteri di accettazione per il prodotto integrato. Il team definisce inoltre come i risultati dei test, come le informazioni sulla copertura, saranno revisionati e comunicati per supportare la valutazione dell'idoneità al rilascio.

3.1.2 Strategia di test del progetto

Una strategia di test del progetto è determinata da diversi fattori contestuali, ciascuno dei quali influenza il modo in cui le tecniche di test, i livelli di test, gli approcci del test e i tipi di test vengono assegnati ai quadranti del testing. La complessità del dominio del prodotto e la criticità del business costituiscono la base della strategia di test. In domini con implicazioni legali o di sicurezza (ad esempio, finanza, sanità, aviazione), vi è una maggiore enfasi sui metodi dei Quadranti 2 e 4, dove sono preferiti approcci tracciabili e strutturati per fornire prove verificabili di conformità e copertura (ad esempio, testing non-funzionale come performance testing, testing di sicurezza, testing di compatibilità o testing di affidabilità).

Struttura organizzativa e composizione del team

Influenzano la profondità e l'ampiezza del testing. Un team interfunzionale con competenze di test integrate in UX, sicurezza o performance consente test più approfonditi in tutti i quadranti. Al contrario, i team che si affidano a esperti esterni o che operano in compartimenti separati potrebbero sottoutilizzare il testing esplorativo e il testing non-funzionale, a meno che non siano esplicitamente pianificati e supportati. Allo stesso modo, quando i team adottano una forte mentalità di responsabilità della qualità da parte dell'intero team, le tecniche di test basate sull'esperienza, come il testing esplorativo del Quadrante 3, vengono applicate continuamente e integrate nello sviluppo piuttosto che limitate alla fine delle iterazioni.

Frequenza di consegna e strategia di release

Determinano il tempo a disposizione per il testing e influenzano direttamente il grado di test automation. I cicli di release brevi e i modelli di continuous delivery richiedono un'ampia test automation nei Quadranti 1 e 4. In questo caso, la scelta dei metodi tende verso opzioni di feedback mantenibili e a basso costo (ad esempio, testing di componente, testing delle API, smoke testing). Al contrario, nelle release a ciclo più lungo o basate su milestone, potrebbero esserci maggiori opportunità di eseguire test manuali nel Quadrante 3 (ad esempio, testing esplorativo, testing di usabilità approfondito e user acceptance testing).

Osservabilità

Se l'osservabilità è bassa, affidarsi al monitoraggio passivo implicito diventa difficile, aumentando la necessità di elementi del Quadrante 2 come simulazioni e prototipi. Se la testabilità è elevata, i team possono introdurre strumentazione e interfacce dedicate esclusivamente ai test per consentire test più mirati in tutti i quadranti.

Maturità del prodotto e frequenza delle modifiche

Nei prodotti in fase iniziale o nei sistemi sottoposti a prototipazione rapida, la strategia di test tende a privilegiare meccanismi di feedback ad alto valore e con un ridotto carico operativo. Il testing esplorativo del Quadrante 3 ha la precedenza sulla modellazione formale o sulla copertura esaustiva. Man mano che il prodotto matura, il rischio di regressione aumenta, richiedendo maggiori investimenti in una test automation mantenibile e nel testing strutturato (ad esempio, testing di componente del Quadrante 1 o testing funzionale del Quadrante 2 progettati con tecniche di test black-box) per mitigare il costo crescente delle modifiche. I Quadranti 1 e 2 dipendono sempre più da tecniche di test basate sulla copertura e da una test automation data-driven per sostenere la velocità.

Limiti degli strumenti e maturità dell'infrastruttura

Laddove sono disponibili moderne pipeline di CI/CD, test double e ambienti containerizzati, la test automation e la virtualizzazione possono essere utilizzate appieno utilizzando i Quadranti 1 e 2, mentre gli strumenti consentono anche l'applicazione di testing non-funzionale dal Quadrante 4. Senza queste capacità, la strategia di test potrebbe essere limitata al testing manuale (ad esempio, testing esplorativo o user acceptance testing dal Quadrante 3) o fare affidamento su ambienti di test isolati. Ciò introduce un rischio e richiede misure compensative quali una maggiore copertura esplorativa e feature toggle per gestire l'incertezza in produzione.

Propensione al rischio e cultura della leadership

Un'organizzazione avversa al rischio o guidata da normative può imporre approcci del test basati su processi complessi, dando priorità alla copertura documentata, a condizioni di test estese e a metriche quantitative. I team in questi contesti tendono ad applicare tecniche di test black-box per il testing funzionale del Quadrante 2 (ad esempio, analisi ai valori limite, testing delle transizioni di stato, testing della tabella delle decisioni). Una cultura che valorizza l'apprendimento adattivo e la sperimentazione supporta il testing esplorativo e lo user acceptance testing del Quadrante 3, nonché le simulazioni del Quadrante 2. Un testing efficace richiede un equilibrio tra tutti gli aspetti: i Quadranti 1 e 2 garantiscono in genere ripetibilità e affidabilità, mentre i Quadranti 3 e 4 favoriscono l'innovazione e l'apprendimento.

Meccanismi di feedback dei clienti e osservabilità della produzione

Possono estendere la strategia di test oltre la consegna. Approcci del test come il testing A/B, la telemetria sull'utilizzo da parte dei clienti e l'analisi dei modelli di failure supportano la validazione continua del valore di business e della tolleranza ai guasti operativi. Queste pratiche di test estendono i Quadranti 3 e 4 agli ambienti di produzione, consentendo cicli di feedback brevi.

La strategia di test dovrebbe essere rivista periodicamente, ad esempio in base ai risultati delle retrospettive. Man mano che il prodotto e il suo contesto si evolvono, la strategia di test dovrebbe adattare il proprio focus spostandosi tra i quadranti, riequilibrando continuamente l'insieme di testing scriptato e non scriptato, nonché di testing manuale e automatizzato. Quanto più dinamico e continuo è il modello di consegna, tanto più spesso si rende necessaria questa rivalutazione.

3.2 Monitoraggio dei Test e Controllo dei Test

Nello sviluppo software Agile, il monitoraggio dei test utilizza metriche leggere visualizzate tramite burn-down chart, flow diagram cumulativi, dashboard automatizzate e visualizzazioni simili per tracciare continuamente lo stato di avanzamento dei test e la qualità del prodotto. Queste visualizzazioni vengono aggiornate frequentemente e sono accessibili a tutti i membri del team per consentire una consapevolezza condivisa della situazione e un'azione immediata quando si verificano deviazioni. In genere, le metriche si focalizzano sul valore fornito e sul fornire un feedback utilizzabile e in tempo reale piuttosto che limitarsi a report periodici. L'enfasi è posta sulla comprensione delle tendenze nell'individuazione dei difetti, nella copertura della test automation, nei risultati del testing esplorativo e nei dati di monitoraggio della produzione, piuttosto che sul tracciamento della conformità con test plan predefiniti.

Nei team Agile, il controllo dei test viene esercitato in modo collaborativo durante gli stand-up quotidiani, le revisioni delle iterazioni e le retrospettive, dove i dati provenienti dalle pipeline di CI/CD guidano le decisioni. Gli aggiustamenti vengono effettuati in tempo reale, spesso a livello di user story o di funzionalità, inclusa la definizione delle priorità delle sessioni di testing esplorativo, il perfezionamento dei criteri di accettazione e la focalizzazione temporanea sulle aree ad alto rischio. Queste attività sono svolte

dall'intero team, non solo dai tester, garantendo che le modifiche all'ambito, alla schedulazione e agli obiettivi di qualità siano discusse e concordate immediatamente. Il feedback della test automation, i segnali di monitoraggio della produzione e i feedback dei clienti sono considerati fattori critici per le decisioni relative al controllo dei test.

Il monitoraggio dei test e il controllo dei test dovrebbero essere integrati nel flusso di lavoro quotidiano, utilizzando la collaborazione interfunzionale e il feedback continuo sia dagli ambienti di pre-produzione che da quelli di produzione. Le decisioni vengono frequentemente rivalutate in base all'evoluzione delle priorità di business e allo stato attuale degli incrementi consegnati, a sostegno del principio Agile di rispondere rapidamente al cambiamento pur continuando a proteggere gli obiettivi di qualità.

3.3 Reporting del Test

Il reporting del test include varie attività, ma questa sezione si focalizza su varie metriche di copertura per riflettere l'avanzamento dei test, supportare il processo decisionale e migliorare la trasparenza. Questi tipi di copertura dovrebbero allinearsi agli obiettivi del team, supportare cicli di feedback rapidi ed essere sufficientemente snelli da risultare utili senza causare un sovraccarico inutile. I team Agile preferiscono indicatori di copertura che favoriscano una comprensione condivisa della qualità e dei rischi del sistema.

La copertura dei requisiti è comune, specialmente nel contesto di ATDD o BDD. Collegando i test di accettazione automatizzati alle user story o agli esempi, i team possono dimostrare quali aspetti funzionali sono stati specificati, sviluppati e verificati. Questi criteri di accettazione diventano unità di copertura tracciabili, aiutando il team a comunicare la readiness e la completezza delle user story.

La copertura del codice può essere utilizzata in modo selettivo, specialmente in ambienti di test altamente automatizzati. Dovrebbe essere interpretata con cautela e non dovrebbe essere considerata un proxy della qualità. Sebbene sia utile per identificare percorsi logici non testati, un'elevata copertura del codice di per sé non garantisce l'efficacia del testing. I team Agile possono combinare questa metrica con la disseminazione dei guasti o con revisioni della qualità delle affermazioni per ottenere una comprensione più approfondita dell'efficacia dei test.

La copertura relativa al testing esplorativo può essere riportata utilizzando il Session-Based Test Management o i tassi di completamento delle test charter. Ciò è particolarmente utile per il testing esplorativo orientato al business, il testing di usabilità o il testing in aree di rischio prive di specifiche formali. È possibile utilizzare dashboard visive per rappresentare ciò che è stato esplorato e quali scenari o personas sono stati utilizzati.

La copertura dell'infrastruttura e dell'ambiente, in particolare nei contesti DevOps e di continuous delivery, può essere tracciata per mostrare quali piattaforme, configurazioni o API sono state esercitate. Gli strumenti di monitoraggio e osservabilità migliorano inoltre il reporting sulla copertura operativa rivelando modelli di utilizzo e anomalie in produzione.

In tutti i casi, le metriche di copertura in Agile dovrebbero essere discusse in modo collaborativo, non solo utilizzate come indicatori astratti. Devono supportare un processo decisionale informato e adattarsi ai contesti organizzativi, di progetto e di team, piuttosto che essere imposte come metriche prescrittive. Le metriche di test sono strumenti di comunicazione e miglioramento nei cicli di vita dello sviluppo software iterativi e adattivi. I team dovrebbero resistere all'illusione di completezza creata dalle metriche e fornire invece informazioni su misura per soddisfare le esigenze dei diversi stakeholder.

3.4 Test Process Improvement

3.4.1 Metriche per il Test Process Improvement

Il test process improvement basato su metriche nello sviluppo software Agile richiede un'attenta interpretazione sia degli indicatori di prodotto che di processo, allineandosi ai principi di miglioramento continuo e attingendo alle capacità e agli strumenti del team. Ad eccezione dei veri e propri key performance indicator, le metriche dovrebbero fungere da meccanismi di feedback piuttosto che da obiettivi di performance, per evitare un uso improprio che potrebbe distorcere il comportamento e portare a un'ottimizzazione locale.

Quando la Defect Detection Percentage diminuisce in modo inaspettato, una root cause analysis può rivelare un testing esplorativo inefficace o una test automation insufficiente. In risposta, il team può prendere in considerazione la possibilità di perfezionare le proprie tecniche di test chartering, migliorare il Session-Based Test Management o investire in strumenti per l'osservabilità o il supporto al testing esplorativo.

Se i test falliscono ripetutamente a causa dell'instabilità dell'ambiente, investire nella gestione dell'ambiente di test, nella virtualizzazione del servizio o in soluzioni containerizzate può migliorare la stabilità e la fiducia nei risultati dei test.

Tempi di ciclo elevati per la risoluzione dei difetti potrebbero indicare una collaborazione debole tra i ruoli o responsabilità compartimentate. Un possibile miglioramento potrebbe comportare l'adozione di whole team approach per la classificazione dei difetti, il coinvolgimento dei tester nelle prime fasi delle sessioni di perfezionamento e l'integrazione di test leader Agile per rafforzare la responsabilità condivisa della qualità. L'abbinamento di metriche quali il mean time to failure e il mean time to repair con le retrospettive consente ai team di riconoscere i colli di bottiglia nei cicli di feedback e di adattarsi di conseguenza.

Una bassa copertura sui percorsi critici, come segnalato dagli strumenti di copertura del codice, può richiedere sforzi per rafforzare la copertura automatizzata a livello di testing di componente o di testing di integrazione dei componenti, riequilibrando potenzialmente la piramide di test verso test più granulari e veloci da eseguire. Tuttavia, i team devono valutare se questi test forniscono informazioni significative o semplicemente gonfiano le metriche.

I test ridondanti o fragili, indicati da alti tassi di failure con bassa resa di difetti, dovrebbero essere rifattorizzati o sostituiti utilizzando modelli di progettazione come i page object o i builder di dati di test. Per ulteriori informazioni sui modelli di progettazione nella test automation, vedere (ISTQB-TAE, v2.0).

Una partecipazione in calo alle retrospettive o alle discussioni sulla qualità può riflettere un problema culturale. Potrebbe essere necessario un coaching per facilitare la sicurezza psicologica e rafforzare il testing come responsabilità condivisa. In un contesto del genere, le metriche qualitative, come la fiducia percepita e l'apertura nella collaborazione, possono essere importanti tanto quanto gli indicatori numerici.

Quando il debito di test si accumula, evidenziato ad esempio da un backlog crescente di flaky test (test instabili) o di tempo dedicato al regression testing manuale, una strategia di test può prevedere tempo in ogni iterazione per la manutenzione dei test, la rotazione della responsabilità dei test e l'applicazione di pratiche di refactoring della test automation. La misurazione dell'efficacia della test automation dovrebbe includere non solo i tassi di superamento/fallimento, ma anche l'impegno richiesto per la manutenzione e i tempi di esecuzione dei test.

Metriche basate sul monitoraggio, come il mean time to recovery o il tasso di difetti sfuggiti ai test e rilevati in produzione, potrebbero dare il via a test process improvements. Un MTTR elevato può indicare la mancanza di avvisi di sistema, vincoli di risorse o testing esplorativo insufficiente dei comportamenti operativi. Ciò potrebbe portare all'integrazione di chaos testing o tecniche di sviluppo software come le canary release, i feature toggle o i dark launch.

Per lo sviluppo delle persone, metriche quali la frequenza del pair testing, il numero di sessioni di apprendimento sui difetti o i rapporti di contributo interfunzionale possono indicare se si stanno verificando la condivisione delle conoscenze e la crescita delle capacità. Una mancanza di collaborazione interfunzionale tra i ruoli può richiedere il mentoring, la rotazione dei ruoli o percorsi di apprendimento strutturati come le iniziative di community of practice.

I miglioramenti relativi agli strumenti possono inoltre essere orientati da metriche quali il tempo che intercorre tra il commit del codice e la ricezione del feedback, l'analisi dei test instabili e il tasso di fallimento delle integrazioni. Strumenti che consentono cicli di feedback brevi, come dashboard, root cause analysis automatizzata o selezione intelligente dei test, possono supportare direttamente l'apprendimento del team e consentire risposte più rapide.

Qualsiasi azione di miglioramento dovrebbe essere misurabile e sottoposta a validazione. A tal fine, le retrospettive possono svolgere un ruolo fondamentale. Le metriche devono essere riconducibili agli obiettivi del test e alle priorità di rischio, piuttosto che a benchmark arbitrari. Modelli come il GQM (Goal-Question-Metric) sono utili per strutturare l'uso delle metriche in modo orientato agli obiettivi. Infine, i team dovrebbero mantenere un pensiero critico e una sensibilità contestuale per evitare che le metriche si distacchino dalla mission del test principale.

3.4.2 Test Process Improvement nello sviluppo software Agile

Per il test process improvement nello sviluppo software Agile, i team possono adottare approcci strutturati ma leggeri che enfatizzano la collaborazione, la consapevolezza del contesto e la sperimentazione iterativa:

Condurre retrospettive mirate. Utilizzare pratiche leggere e collaborative, come le retrospettive specifiche per i test, per identificare le sfide dei test e le opportunità di miglioramento. L'effort di miglioramento deve essere radicato nel contesto attuale del team, non imposto dall'esterno. Le retrospettive incentrate specificamente sui test, distinte dalle retrospettive Agile generali, possono fornire una visione mirata dell'efficacia delle pratiche di testing e del loro allineamento con le esigenze di business e tecniche (ISTQB-ATLAS, v1.0), e aiutare a costruire una responsabilità condivisa degli obiettivi di qualità.

Stabilire una valutazione di base. Eseguire valutazioni strutturate dei test Agile per identificare le lacune nell'attuale approccio del test. Strumenti come le autovalutazioni o le checklist possono stimolare la riflessione sulle dimensioni fondamentali del testing Agile, quali la collaborazione, i cicli di feedback e la continuous integration. Da lì, il miglioramento mirato può seguire un percorso graduale e iterativo.

Allinearsi tramite workshop congiunti sui rischi. Coinvolgere rappresentanti del business, del testing e dello sviluppo per costruire una comprensione condivisa dei rischi di prodotto critici e di dove focalizzare al meglio l'effort di test. Ciò garantisce che i miglioramenti supportino sia gli obiettivi di business che quelli del team.

Rafforzare i cicli di feedback di CI/CD. Integrare pratiche come gli approcci test-first e il pair testing per consentire un feedback tempestivo e concreto. Utilizzare il testing esplorativo non solo per individuare i

difetti, ma anche per scoprire rischi sconosciuti e lacune nei test, supportando il perfezionamento adattivo del processo di test.

Eseguire piccoli esperimenti. Adottare tecniche, come il Session-Based Test Management, per esplorare i cambiamenti del processo di test in modo incrementale. I risultati di questi esperimenti dovrebbero essere discussi apertamente all'interno del team, utilizzando retrospettive per adeguare la strategia di test secondo necessità.

Utilizzare le metriche visibili con cautela. Monitorare lo stato di salute della test automation, le tendenze dei difetti e la copertura nelle pipeline di CI/CD, ma resistere alla tentazione di mettere a confronto i team tra loro. Focalizzarsi invece su metriche che aiutino i team a comprendere la propria efficacia e a rendere il loro lavoro più visibile.

I team possono utilizzare pratiche quali la Value Stream Mapping, il ciclo PDCA o i casual loop diagram (vedere (ISTQB-ATLAS, v1.0)). I miglioramenti devono tenere conto di fattori sociotecnici, quali la cultura organizzativa e le pratiche di comunicazione. La resistenza al cambiamento (vedere: (Prosci, 2026)) dovrebbe essere affrontata attraverso conversazioni inclusive che attingano a strategie di facilitazione del cambiamento.

Soprattutto, il miglioramento continuo nel testing Agile deve essere radicato nel pensiero critico, nell'apprendimento continuo e nella collaborazione di tutto il team. L'applicazione di modelli o processi fissi dovrebbe essere adattata con attenzione al contesto. Ogni miglioramento dovrebbe fungere da ipotesi da testare attraverso l'azione, l'apprendimento e l'adattamento.

4 Shift-Left - 135 minuti

Parole Chiave

Acceptance Test-Driven Development, Behavior-Driven Development, qualità, shift-left, base di test, testboarding, testware

Parole Chiave Specifiche del Dominio

bias, example mapping, specifica per esempi, storyboarding, conversazione strutturata, scheda della sessione di test, user journey, user story, user story slicing

Obiettivi di Apprendimento:

4.1 Utilizzare lo Shift-Left per Migliorare la Qualità della Base di Test

CTAL-AT-4.1.1 (K2) Fornire esempi di come il testware possa essere utilizzato come una forma di requisiti

CTAL-AT-4.1.2 (K2) Spiegare come lo storyboarding e il testboarding possano essere utilizzati per aumentare la qualità della base di test

CTAL-AT-4.1.3 (K2) Spiegare come l'example mapping possa essere utilizzato per aumentare la qualità della base di test

CTAL-AT-4.1.4 (K2) Fornire esempi di come i bias possano influire negativamente sulla qualità del prodotto

CTAL-AT-4.1.5 (K3) Applicare lo user story slicing per ottenere user story testabili

4.2 Shift-Left e Requirements Engineering

CTAL-AT-4.2.1 (K2) Spiegare in che modo il requirements engineering supporta lo shift-left

Introduzione

Lo shift-left pone l'accento sull'introduzione di attività di qualità il più presto possibile nel ciclo di vita dello sviluppo software, focalizzandosi sul miglioramento della base di test fin dall'inizio. Il testware, come i criteri di accettazione, le checklist e i test script automatizzati, possono fungere da requisiti dinamici o supportare e documentare i requisiti, garantendo chiarezza e comprensione condivisa. Tecniche come lo storyboarding e il testboarding aiutano a visualizzare i flussi di lavoro e ad allineare le prospettive, mentre l'example mapping perfeziona i criteri di accettazione attraverso la discussione collaborativa. La consapevolezza dei bias cognitivi protegge da ipotesi errate che potrebbero ridurre la qualità del prodotto. Pratiche come lo user story slicing assicurano che le user story siano piccole, mirate e testabili. Parallelamente, un requirements engineering efficace supporta lo shift-left combinando analisi, documentazione e validazione, e selezionando metodi di elicitazione adeguati, come interviste, workshop o prototipazione, per cogliere le esigenze degli stakeholder. Adottando lo shift-left, il team può rilevare anomalie e difetti prima, ridurre i rework e consegnare prodotti di maggior valore. Gli strumenti emergenti di AI e di large language model possono supportare lo shift-left analizzando i requisiti alla ricerca di ambiguità, suggerendo scenari di test dai criteri di accettazione o identificando lacune nella copertura delle user story. I tester dovrebbero valutare tali strumenti in modo critico, comprenderne i limiti e utilizzarli per potenziare, piuttosto che sostituire, l'analisi collaborativa umana.

4.1 Utilizzare lo Shift-Left per Migliorare la Qualità della Base di Test

4.1.1 Testware come requisiti

Nello sviluppo software Agile, il testware come le user story con criteri di accettazione, le checklist, gli scenari BDD, i test case a livello di accettazione, gli esempi, le test charter, i contratti o i test script automatizzati può essere trattato come una forma di requisiti quando utilizzato in modo collaborativo, e revisionato e applicato per ogni iterazione. Questi prodotti di lavoro aiutano i team a chiarire le aspettative, a promuovere una comprensione condivisa e a supportare la validazione continua del comportamento del software. Sono spesso creati durante le conversazioni con gli stakeholder di business, rendendoli requisiti dinamici piuttosto che documentazione statica.

I requisiti utilizzati nel Behavior-Driven Development (BDD), nell'Acceptance Test-Driven Development (ATDD) o nella specifica per esempi (Adzic, 2011) sono derivati dalle conversazioni tra il team e gli stakeholder di business e sono spesso documentati in un formato che può essere eseguito come test. Queste specifiche eseguibili incarnano i criteri di accettazione fungendo sia da requisiti dinamici che da regression test duraturi.

Nel testing esplorativo, le test charter session-based spesso emergono da idee di prodotto di alto livello o da valutazioni del rischio. Queste test charter guidano l'indagine e aiutano a identificare funzionalità mancanti o fraintese. Le schede delle sessioni di test generate durante la sessione di testing esplorativo forniscono approfondimenti informali ma preziosi sui requisiti in evoluzione.

Nei modelli DevOps e di continuous delivery, i log provenienti dal monitoraggio, dai dati analitici dei clienti e dai cicli di feedback possono fungere da requisiti in evoluzione, quando interpretati e tradotti in ipotesi testabili o regole decisionali, specialmente quando il cambiamento è guidato dal comportamento in produzione e dai dati di interazione degli utenti. I team possono catturare queste informazioni implicite in test case o checklist per una validazione continua.

Altre forme di testware, come mappe mentali, checklist e modelli visivi (ad esempio, flow diagram o state diagram) possono essere utilizzate durante la pianificazione Agile e possono guidare direttamente il lavoro di sviluppo. Esse traducono la comprensione implicita in punti di riferimento condivisi.

4.1.2 Storyboarding e Testboarding

Lo storyboarding e il testboarding sono tecniche collaborative che migliorano la qualità della base di test potenziando la comprensione condivisa e l'individuazione precoce di ambiguità, rischi e omissioni. Dal punto di vista del testing del software, sono particolarmente efficaci nei team Agile focalizzati sul feedback e sulla collaborazione dell'intero team.

Lo storyboarding visualizza gli user journey presentando il comportamento previsto come una sequenza di passaggi o scene (Walker et al., 2013). Questo approccio narrativo rivela ipotesi, edge case e interazioni che potrebbero non essere evidenti nei requisiti testuali. Suscita domande critiche sulla logica del flusso di lavoro, consentendo agli stakeholder di individuare incongruenze o flussi incompleti prima che il codice venga scritto. Poiché gli storyboard si focalizzano sulle interazioni degli utenti, aiutano a scoprire lacune nella base di test relative all'usabilità, alle transizioni di flusso e alle esigenze specifiche dei persona.

Il testboarding è una pratica semplice che aiuta a visualizzare ciò che deve essere testato per ogni funzionalità, completando lo storyboarding. Oltre a validare i criteri di accettazione, il testboarding mostra gli scenari principali e alternativi, il che porta a un allineamento precoce tra tester, sviluppatori e rappresentanti del business su terminologia, comportamento del sistema e priorità di test. Questa comprensione condivisa riduce gli errori di interpretazione durante la progettazione dei test e l'esecuzione dei test, aumentando così la manutenibilità dei test case.

Combinando queste tecniche, i team creano uno spazio condiviso per il pensiero esplorativo. I tester possono porre domande del tipo "cosa succederebbe se?" e proporre scenari negativi o alternativi, stimolando miglioramenti precoci nella user story e nei suoi criteri di accettazione. Le pratiche che aiutano a esplorare precocemente la base di test migliorano l'identificazione del rischio e riducono la rilevazione tardiva di difetti critici, consentendo ai tester di stabilire correttamente le priorità dei test.

Queste tecniche sono più efficaci se utilizzate in modo iterativo: riesaminare gli storyboard e i testboard durante il perfezionamento del backlog e la pianificazione delle iterazioni mette in luce i cambiamenti nell'ambito, nei rischi o nella comprensione. Possono fungere da documentazione dinamica che si evolve insieme al prodotto e alla base di test, aiutando i tester a rilevare quando i requisiti si discostano o gli obiettivi di qualità sono compromessi.

4.1.3 Example Mapping

L'example mapping (Wynne, 2015) è una tecnica collaborativa che perfeziona le user story attraverso una conversazione strutturata. Aiuta i team a identificare le regole di business, gli esempi associati (che possono diventare test case) e a raccogliere domande per approfondire la comprensione condivisa. Il team crea delle card colorate per rappresentare i diversi elementi della discussione (ad esempio, giallo per la user story in fase di review, blu per le regole di business, verde per gli esempi che illustrano regole e rosso per le domande). Le regole sono raggruppate sotto una user story e gli esempi sono raggruppati sotto le regole corrispondenti. Il team continua a creare card finché non è soddisfatto della chiarezza dell'ambito della user story o finché non scade il tempo limite. Quando il tempo scade, può indicare che la user story è troppo complessa o necessita di essere perfezionata e il team decide i passi successivi.

Man mano che la conversazione procede, il team costruisce rapidamente una rappresentazione visiva della user story che riflette la sua attuale comprensione condivisa. Troppe card "domanda" suggeriscono che il team ha ancora molto da imparare su questa user story. Un gran numero di card "regola", o una singola regola con molte card "esempio", suggeriscono che la user story è ampia e complessa e dovrebbe essere suddivisa (vedere *Sezione 4.1.5*). In tali casi, il team divide le regole e gli esempi tra la story originale e la story risultante dalla suddivisione, crea un'altra card gialla (user story) per la user story suddivisa e la inserisce nel backlog.

Dal punto di vista del testing, l'example mapping rafforza la base di test ancorando i requisiti astratti a esempi concreti e concordati. Questi esempi riducono l'ambiguità, portano alla luce presupposti nascosti e rivelano tempestivamente interpretazioni contrastanti.

Nello sviluppo software Agile, la base di test spesso si evolve durante le fasi di perfezionamento e implementazione. L'example mapping migliora la qualità garantendo una comprensione condivisa da parte dell'intero team e chiarendo i criteri di accettazione che guidano direttamente Acceptance Test-Driven Development (ATDD) e Behavior-Driven Development (BDD). Promuove la validazione dello shift-left individuando le lacune nelle specifiche prima dell'inizio della codifica.

L'example mapping supporta la creazione di test charter per il testing esplorativo e aiuta i tester a identificare condizioni di test rilevanti e scenari alternativi, poiché gli esempi concreti e le domande suggeriscono naturalmente i limiti testabili e gli edge case da esplorare. Se utilizzata in modo iterativo, rende la base di test più resiliente al cambiamento, poiché la tracciabilità dagli esempi alle regole e alle esigenze degli utenti viene preservata durante tutto lo sviluppo.

4.1.4 Bias nel testing Agile

Un bias è una deviazione sistematica dalla neutralità o dall'accuratezza che influenza il modo in cui le persone pensano, giudicano o agiscono, talvolta allontanandole dall'obiettività o dall'equità. I bias possono compromettere significativamente la qualità del prodotto distorcendo la percezione del rischio, limitando la diversità del testing e indebolendo i meccanismi di feedback fondamentali per le pratiche di sviluppo software Agile. Nel testing, i bias possono avere origine da individui, team o dalla cultura organizzativa e influenzare le decisioni inconsciamente.

I bias particolarmente rilevanti per il testing Agile includono:

- Il bias di conferma può portare a progettare ed eseguire test che verificano solo il comportamento previsto, trascurando potenziali percorsi di failure o condizioni insolite. Ciò restringe l'ambito di esplorazione e aumenta il rischio di difetti non rilevati, specialmente in sistemi complessi in cui le decisioni di sviluppo sono guidate da presupposti impliciti.
- L'effetto di ancoraggio si verifica quando i requisiti iniziali o le aspettative degli stakeholder fissano i team su interpretazioni specifiche, trascurando comportamenti alternativi. Ad esempio, se le user story iniziali si focalizzano esclusivamente sui percorsi positivi, i test potrebbero sottorappresentare gli edge case, l'accessibilità o le preoccupazioni relative all'internazionalizzazione.
- Il bias di conformità può compromettere la qualità quando le opinioni diverse vengono scoraggiate, i tester copiano i comportamenti o le convinzioni gli uni degli altri invece di seguire il proprio giudizio indipendente, oppure i tester si rimettono alle voci dominanti nei team interfunzionali. Ciò mina il valore del whole team approach, che si basa sulla responsabilità condivisa per la qualità e sui cicli di feedback critici durante tutto il ciclo di vita dello sviluppo software.

I bias possono influenzare la progettazione dei criteri di accettazione, riducendo la chiarezza e aumentando l'ambiguità nei test di accettazione automatizzati o nelle specifiche eseguibili. In pratiche

come la specifica per esempi e l'ATDD, il mancato mettere in discussione le ipotesi durante la specifica collaborativa può portare a una copertura inefficace.

Il rischio di bias è aumentato dal whole team approach. Per mitigare i bias, i team dovrebbero promuovere la diversità di pensiero, utilizzare il testing esplorativo con il testing session-based e promuovere una cultura di sicurezza psicologica che incoraggi il dissenso e la messa in discussione. Pratiche come i debriefing dopo le sessioni di testing esplorativo, il pair testing, l'uso di tissue tester e la root cause analysis possono aiutare a rivelare l'influenza dei bias e rafforzare la consapevolezza collettiva.

4.1.5 User Story Slicing

Lo user story slicing è la pratica di suddividere le user story di grandi dimensioni in parti più piccole e gestibili ("slice") che apportano valore, sono fattibili all'interno di un'iterazione e sono testabili. Le user story correttamente suddivise facilitano la continuous delivery e consentono un feedback più tempestivo, supportando così i principi Agile.

Per applicare lo user story slicing in modo da supportare la testabilità, il team inizia identificando le diverse dimensioni lungo le quali una user story può essere suddivisa:

- **La suddivisione del flusso di lavoro** scompone la user story in base alle attività dell'utente o alle fasi di un processo di business. Ogni slice dovrebbe focalizzarsi su una singola attività o interazione, idealmente una che dia luogo a un comportamento osservabile del sistema. Ciò rende più facile definire criteri di accettazione precisi e risultati attesi.
- **La suddivisione in base alla complessità dei dati** inizia con un singolo esempio di percorso positivo che utilizza i dati più semplici e sposta le condizioni alternative dei dati e gli edge case in user story separate. Ciò consente al team di verificare le funzionalità principali in anticipo e di aumentare la copertura in modo incrementale. Ogni slice dovrebbe includere una variazione sufficiente a consentire test significativi senza sovraccaricare l'ambito.
- **La suddivisione per interfaccia** separa la logica di elaborazione del backend dalle funzionalità della user interface (UI) quando ognuna di esse può essere testata in modo indipendente. Ciò consente una test automation più precoce e si allinea alla stratificazione architettonica, specialmente in ambienti di continuous integration. Il team costruisce user story del backend che espongono gli output tramite API prima di lavorare su user story che includono flussi di lavoro della UI.
- **La suddivisione per scenario** parte dallo scenario di business più comune e si sviluppa su di esso creando ulteriori slice per casi meno frequenti o più complessi. Il team può utilizzare l'example mapping o conversazioni strutturate per elicitare questi scenari durante il perfezionamento. Ogni parte di scenario dovrebbe essere verificabile attraverso esempi specifici che costituiscono la base per i test orientati al business.
- **La suddivisione verticale** garantisce che ogni user story includa una slice limitata ma completa di funzionalità attraverso i livelli, dal database alla UI, ove appropriato. Questo aiuta i team a fornire un valore dimostrabile, eseguire testing end-to-end e verificare l'integrazione tra i componenti. Tuttavia, si dovrebbe evitare di includere troppo in una singola slice. Se l'ambito end-to-end diventa ingestibile, il team dovrebbe tornare alla suddivisione per scenario o del flusso di lavoro.
- **La suddivisione basata sui vincoli** si focalizza prima sulle ipotesi generali e poi considera le condizioni derivate dai vincoli noti. Ad esempio, il team parte da una user story che presuppone che l'utente sia autenticato, quindi ne ricava una user story separata per gestire i failure

dell'autenticazione. Questo aiuta a isolare le condizioni di test e migliora la chiarezza nella definizione dei limiti.

I criteri di accettazione spesso determinano il successo o il fallimento dello user story slicing. Durante la suddivisione, il team non può limitarsi a suddividere la user story; deve ripensare e riscrivere i criteri di accettazione in modo che ogni slice abbia senso. Ad esempio, le user story di grandi dimensioni hanno solitamente una serie di criteri di accettazione generici che dovrebbero essere perfezionati dopo la suddivisione per adattarsi all'ambito più ristretto. I criteri di accettazione dovrebbero focalizzarsi su un singolo risultato specifico per l'utente per ogni slice di user story, evitando di combinare più variazioni comportamentali in una singola slice. Durante la suddivisione, i criteri di accettazione originali non dovrebbero essere copiati e incollati in ogni slice. Dovrebbero invece essere riscritti per ogni slice in modo che ognuna sia testabile, possa essere dimostrata e corrisponda alla propria slice. L'insieme dei criteri di accettazione tra le slice alla fine dovrebbe coprire l'intera story originale.

Indipendentemente dalla strategia di suddivisione, il team dovrebbe sempre considerare la testabilità come un criterio primario. Ciò avviene chiedendosi se la slice può essere validata in modo indipendente, se è possibile ricavarne criteri di accettazione chiari e se supporta la creazione di test orientati al business. Se una qualsiasi di queste condizioni non è soddisfatta, la user story necessita di un ulteriore perfezionamento.

4.2 Shift-Left e Requirements Engineering

Lo shift-left sposta le attività di qualità a una fase più precoce del ciclo di vita dello sviluppo software per individuare tempestivamente anomalie e difetti, ridurre i rework, accorciare i cicli di consegna e migliorare l'allineamento con le esigenze di business. Tutte le principali attività del requirements engineering – elicitazione, analisi, specifica e validazione dei requisiti – possono supportare lo shift-left riducendo i malintesi, prevenendo i difetti e minimizzando costosi rework nelle fasi successive del ciclo di vita dello sviluppo software:

- L'elicitazione dei requisiti supporta lo shift-left catturando le esigenze degli stakeholder in modo precoce e continuo, consentendo allo sviluppo e al testing di iniziare con una chiara comprensione delle aspettative degli stakeholder. L'elicitazione continua attraverso user story, backlog grooming e workshop collaborativi assicura che le funzionalità siano definite, testabili e validate, riducendo i difetti e i rework nelle fasi successive dell'iterazione.
- L'analisi dei requisiti supporta lo shift-left identificando tempestivamente ambiguità, conflitti, rischi e dipendenze, garantendo che lo sviluppo e il testing inizino con requisiti chiari, fattibili e prioritizzati. Analizzando i requisiti, i team possono progettare soluzioni appropriate, pianificare i test e prevenire i difetti prima dell'inizio della codifica, riducendo costosi rework nelle fasi successive del ciclo di vita dello sviluppo software.
- La specifica dei requisiti supporta lo shift-left fornendo una documentazione chiara, precisa e testabile di ciò che il sistema deve fare, consentendo a sviluppatori e tester di pianificare e validare il lavoro in anticipo. Requisiti ben specificati riducono i malintesi, guidano la creazione precoce dei test e aiutano a individuare anomalie prima dell'implementazione, minimizzando i difetti a valle e i rework. I tester Agile lavorano tipicamente con rappresentazioni testuali in formato libero come user story ed epic. Quando è necessaria una maggiore struttura, possono utilizzare modelli di scenario semi-formali come use case, business process model (ad es. BPMN) o state transition diagram, a seconda del contesto. I modelli di scenario possono illustrare i flussi di interazione, inclusi gli edge case che potrebbero non essere catturati nelle descrizioni di alto livello. Spesso vengono mantenuti

dei glossari per garantire una comprensione condivisa dei termini specifici del dominio, specialmente quando si definiscono i criteri di accettazione.

La validazione dei requisiti supporta un approccio shift-left assicurando che i requisiti siano corretti, inequivocabili, completi, coerenti e testabili prima dell'inizio dello sviluppo. La validazione precoce attraverso review, prototipazione e modellazione aiuta a individuare anomalie e lacune, riducendo i difetti, i rework e le costose modifiche in fase avanzata.

5 Approcci e Tecniche di Test Agile - 285 minuti

Parole Chiave

sviluppo software Agile, edge case, testing esplorativo, mob testing, pair testing, test case, test charter, euristica di test, mnemonico di test, test smell, tour, vibe testing

Parole Chiave Specifiche del Dominio

epic, FEW HICCUPPS, I SLICED UP FUN, RCRCRC, SFDIPOT, TERMS

Obiettivi di Apprendimento:

5.1 Testing Esplorativo

CTAL-AT-5.1.1 (K2) Spiegare le euristiche di test

CTAL-AT-5.1.2 (K2) Fornire esempi di mnemonici di test relativi al testing nello sviluppo software Agile

CTAL-AT-5.1.3 (K2) Spiegare i tour di test

CTAL-AT-5.1.4 (K4) Analizzare le user story e le epic per creare le test charter

CTAL-AT-5.1.5 (K3) Applicare il testing esplorativo a supporto del testing nello sviluppo software Agile

5.2 Testing Assistito

CTAL-AT-5.2.1 (K2) Spiegare il mob testing

CTAL-AT-5.2.2 (K2) Spiegare il pair testing

CTAL-AT-5.2.3 (K2) Spiegare il vibe testing

5.3 Test Smell

CTAL-AT-5.3.1 (K3) Utilizzare i test smell per valutare la qualità dei test case

Introduzione

I tester devono adattare le loro strategie di test e i loro strumenti per adeguarsi alla natura iterativa e al ritmo serrato dei progetti di sviluppo software Agile. Questo capitolo esplora una serie di tecniche di test e metodi di collaborazione che migliorano l'efficacia e l'efficienza del testing nello sviluppo software Agile. Essi enfatizzano la flessibilità, il feedback in tempo reale e l'apprendimento continuo – valori fondamentali delle metodologie di sviluppo software Agile.

5.1 Testing Esplorativo

5.1.1 Euristiche di test

Le euristiche di test sono linee guida generali o regole empiriche che aiutano i tester a prendere decisioni informate, risolvere problemi e guidare la loro esplorazione di un sistema in fase di test. Non è garantito che siano corrette in ogni contesto, ma fungono da strategie pratiche che spesso portano a risultati preziosi quando:

- il tempo o le risorse sono limitati;
- le specifiche sono incomplete o poco chiare;
- il tester esegue test basati sull'esperienza;
- il regression testing viene eseguito senza una copertura completa della test automation;
- il tester vuole pensare come un utente o simulare use case del mondo reale.

Esempi di euristiche di test includono:

Linee guida – procedure o passaggi raccomandati che indicano ai tester le azioni specifiche da intraprendere o gli approcci da seguire durante il testing. Esempi includono: "testare presto e spesso", "il testing riguarda le informazioni, non solo il superamento o il fallimento", "considerare sempre i limiti", "provare flussi di lavoro comuni e comportamenti inaspettati o non validi", "modificare una variabile alla volta per isolare gli effetti durante il testing di opzioni, configurazioni o input", "testare come le funzionalità funzionano insieme in sequenza, non solo isolatamente".

Checklist generiche – elenchi riutilizzabili di elementi comuni da testare, utilizzati come promemoria o suggerimenti per esplorare aree che altrimenti potrebbero essere trascurate. Le checklist possono focalizzarsi su varie aree, quali caratteristiche di qualità, tipi di test, livelli di test, aree di rischio, ruoli degli utenti e ambienti di test. Esempi includono le euristiche di usabilità di Nielsen (Nielsen, 1994), le OWASP Top 10 dei rischi di sicurezza delle applicazioni web (OWASP, 2025), le tassonomie dei difetti come la tassonomia di Beizer (Beizer, 1990) e la tassonomia IEEE 1044 (*IEEE 1044 – Classification for Software Anomalies*, 2009).

Regole empiriche – "leggi" esperienziali non scritte o verità comunemente accettate. Esempi includono: "se qualcosa sembra troppo semplice, probabilmente nasconde complessità", "se si è rotto prima, probabilmente si romperà di nuovo", "i difetti si raggruppano in cluster", "se è difficile da testare, probabilmente è difficile da usare", "ciò che funziona con un utente potrebbe fallire con molti altri".

Mnemonici – ausili mnemonici che aiutano i tester a richiamare rapidamente una serie di condizioni di test. Esempi includono SFDIPOT (vedere la *Sezione 5.1.2*).

Analogie e metafore – consentono ai tester di confrontare il sistema con concetti familiari e quindi applicare il ragionamento derivante da tali concetti per generare condizioni di test. Esempi includono i test tour (vedere la *Sezione 5.1.3*).

5.1.2 Mnemonici di test

Gli mnemonici sono ausili mnemonici, solitamente sotto forma di acronimi, abbreviazioni, rime o schemi. Gli mnemonici non prescrivono rigidamente cosa testare, ma aiutano i tester a richiamare insieme di informazioni complesse e ad ampliare la loro prospettiva durante il testing esplorativo e le revisioni. Tuttavia, per essere efficaci, richiedono conoscenze del dominio, del contesto e dei rischi, oltre a pensiero critico e fiducia nel team. Nel testing del software, gli mnemonici di test sono comunemente utilizzati per:

- guidare il testing esplorativo fornendo una struttura al testing in forma libera senza test case rigidi;
- stimolare il pensiero creativo e critico, spingendo i tester a ragionare in modi che potrebbero non aver considerato naturalmente;
- ottenere una copertura più ampia assicurando che le aree chiave non vengano dimenticate;
- fungere da checklist mentali durante la pianificazione dei test o l'esecuzione dei test, riducendo il carico cognitivo condensando concetti complessi in formati facili da ricordare.

Il testing del software impiega vari mnemonici di test e i tester possono creare i propri set personalizzati. Alcuni mnemonici di test popolari includono:

- SFDIPOT (Structure, Function, Data, Interfaces, Platform, Operations, Time) – supporta l'analisi generale del sistema in fase di test (Bolton, 2014). Può essere utilizzato quando si esplora un nuovo sistema o si pianificano le test charter.
- RCRCRC (Recent, Core, Risky, Configuration, Repaired, Chronic) – supporta il regression testing, dove ogni lettera rappresenta una parola per aiutare a scoprire le condizioni di test (Johnson, 2012). Può essere utilizzato per decidere la copertura nelle iterazioni di sviluppo software Agile o nei cicli di regression testing.
- I SLICED UP FUN (Inputs, Store, Location, Interaction/Interruptions, Communication, Ergonomics, Data, Usability, Platform, Function, User scenarios, Network) (Kohl, 2010). Può essere utilizzato per supportare il testing delle applicazioni mobile nella generazione delle condizioni di test.
- FEW HICCUPPS (Familiarity, Explainability, World, History, Image, Comparable products, Claims, Users' desires, Product, Purpose, Statuses) – supporta l'identificazione e l'applicazione degli oracoli del test (Bolton, 2012). Può essere utilizzato per identificare i difetti osservando diversi tipi di incongruenze.
- TERMS (Tools and Technology, Execution, Requirements and Risks, Maintenance, Security) (Gareev, 2019). Aiuta a ricordare i diversi fattori che influenzano il successo della test automation.

5.1.3 Tour di test

I tour di test utilizzano metafore ispirate ai tour cittadini per guidare il testing esplorativo di un sistema. L'idea è di aiutare i tester a esplorare il software, proprio come i turisti esplorano un posto nuovo, utilizzando diversi tipi di "tour" per scoprire vari tipi di problemi. Ogni tour si focalizza su una prospettiva specifica, guidando il tester nell'esame dell'applicazione da un punto di vista particolare. I tour sono progettati per essere informali, leggeri e creativi, rendendoli ideali per il testing esplorativo, il testing session-based o come generatori di condizioni di test.

Esempi di tour di test, adattati dai quartieri cittadini (Whittaker, 2009), includono:

Tour del quartiere degli affari (Business district). Per il software, si riferisce alle funzionalità principali. Quest'area è delimitata dal codice di avvio e di spegnimento e contiene le funzionalità e le funzioni con cui gli utenti interagiscono nel software.

Tour del quartiere storico (Historical district). Per il software, si riferisce al codice legacy e alla storia delle funzioni o funzionalità con difetti. Come la storia reale, il codice legacy è spesso poco compreso e vengono fatte molte supposizioni quando il codice legacy viene incluso, modificato o utilizzato.

Tour del quartiere turistico (Tourist district). Molte città hanno quartieri che i turisti visitano principalmente. La gente del posto e i residenti tendono ad evitare queste zone. Il software rispecchia questa situazione, con gli utenti inesperti che gravitano verso funzionalità e funzioni di cui gli utenti esperti non hanno più bisogno.

Tour del quartiere dei divertimenti (Entertainment district). Nelle città, questi quartieri offrono relax e intrattenimento. Anche il software, come le città, ha funzionalità di supporto. Questi tour completano quelli attraverso altri quartieri e colmano le lacune di un test plan completo.

Tour del quartiere degli hotel (Hotel district). Nelle città, questi quartieri sono un luogo di riposo. Nel testing, è un luogo in cui il tester del software può allontanarsi dalle funzionalità primarie e dalle funzionalità più popolari e testare alcune delle funzioni secondarie e di supporto che sono spesso trascurate o sottorappresentate in un test plan.

I **quartieri malfamati** sono pericolosi, ma attraggono comunque una certa categoria di turisti. I tour nei quartieri malfamati sono essenziali per i tester in quanto rivelano aree di vulnerabilità che potrebbero essere altamente problematiche per gli utenti se rimanessero nel prodotto. Quest'area si occupa principalmente di testing negativo e impiega tecniche di test come l'error guessing o gli attacchi basati sui guasti.

5.1.4 Creazione delle test charter

Le test charter sono strettamente allineate ai principi dello sviluppo software Agile e forniscono un approccio strutturato ma flessibile al testing esplorativo (vedere la *Sezione 5.1.5*). Una test charter è una dichiarazione breve e mirata che delinea lo scopo, l'ambito e gli obiettivi del test di una sessione di testing esplorativo. Funge da dichiarazione di intenti guida o ipotesi testabile per il tester durante la sessione di test. Le test charter possono essere creati sulla base di varie fonti, quali user story, criteri di accettazione, rischi di prodotto o obiettivi di iterazione. Come minimo, una test charter contiene:

- Missione/obiettivi del test – focus di alto livello o obiettivi testabili per la sessione (ad esempio, "Validare il login dell'utente in condizioni di carico")
- Ambito – aree specifiche di interesse all'interno del sistema in fase di test, del livello di test, delle condizioni di test, delle tecniche di test da utilizzare, dei criteri di uscita, delle priorità, di cosa coprire e cosa non sarà coperto

e può includere anche altre informazioni, quali:

- Attore – utente previsto del sistema
- Scopo – un obiettivo che l'attore vuole raggiungere
- Configurazione – cosa deve essere predisposto per avviare il testing esplorativo (ad es. ambiente di test)

- Priorità – importanza relativa di questa test charter, basata sulla priorità della user story associata o sul livello di rischio
- Riferimento – specifiche (ad es. user story), rischi o altre fonti di informazioni
- Dati – tutti i dati necessari per eseguire la test charter
- Attività – un elenco di idee su ciò che l'attore potrebbe voler fare con il sistema, cosa sarebbe interessante testare e quali tecniche di test potrebbero essere utilizzate
- Note sull'oracolo del test – come valutare il prodotto per determinare i risultati corretti del test (ad es. per catturare ciò che accade sullo schermo e confrontarlo con quanto scritto nel manuale dell'utente)
- Variazioni – azioni e valutazioni alternative a integrazione delle idee descritte nella sezione delle attività
- Limiti – ad esempio, ciò che il prodotto non deve mai fare
- Vincoli e rischi – ad esempio, normative, regole e standard utilizzati

Per ricavare le test charter dalle user story e dalle epic, i tester Agile le analizzano per identificare gli obiettivi del test e il focus della missione. Estraggono l'intento della user story o dell'epic (ad esempio, chi è l'attore e qual è il suo scopo) e lo traducono in una dichiarazione di intenti per la sessione di test. Le user story correlate possono essere raggruppate per valutare il comportamento integrato.

I tester Agile identificano ciò che deve essere appreso, verificato o esplorato durante la sessione di test. Possono utilizzare la tecnica 5W1H (What, Who, When, Where, Why, How), nota anche come metodo Kipling, per rispondere in modo esaustivo alle domande esistenti e stimolare idee che potrebbero contribuire alla risoluzione di un problema:

- Quale funzionalità viene fornita?
- Chi sono gli utenti?
- Quando viene utilizzata?
- Dove viene utilizzata?
- Perché questa funzionalità è importante?
- Come si integra con le altre funzionalità?

Queste domande aiutano a definire l'ambito, la configurazione e la priorità. Diverse euristiche di test (vedere la *Sezione 5.1.1*, la *Sezione 5.1.2* e la *Sezione 5.1.3*) possono essere utili per rispondere a queste domande e generare condizioni di test, dati di test e attività di test da includere nella test charter.

I criteri di accettazione definiscono ciò che deve verificarsi affinché una user story sia accettata. Questi possono essere tradotti direttamente in condizioni di test. Sebbene i criteri di accettazione guidino il comportamento previsto, evidenziano anche aree da esplorare al di là del "percorso felice": Cosa succede se i criteri di accettazione non vengono soddisfatti? Quali edge case o condizioni di test negative si presentano? I criteri di accettazione sono chiari o potenzialmente ambigui? Questo aiuta i tester a sviluppare condizioni di test sia per i test positivi che per quelli negativi. I criteri di accettazione spesso implicano o specificano requisiti relativi ai dati, come valori di input validi, precondizioni o formati o intervalli specifici.

I criteri di accettazione possono anche fungere da note di oracolo del test leggere perché definiscono i comportamenti attesi, stabiliscono soglie di superamento/fallimento e si allineano alle aspettative dei

rappresentanti del business. Tuttavia, i tester dovrebbero ricordare che i criteri di accettazione non sono oracoli del test esaustivi: non catturano tutti gli aspetti della qualità. Il testing esplorativo è ancora necessario per scoprire difetti al di fuori dell'ambito dei criteri di accettazione. I tester Agile dovrebbero considerare i criteri di accettazione mancanti, vaghi o ambigui come opportunità per il testing esplorativo, al fine di scoprire variazioni non dichiarate e edge case.

5.1.5 Eseguire il testing esplorativo

Negli ambienti di sviluppo software Agile, dove i requisiti cambiano frequentemente, il testing esplorativo offre la flessibilità e la reattività che spesso mancano agli approcci guidati dagli script. Il testing esplorativo è importante anche nei progetti di sviluppo software Agile a causa del tempo limitato a disposizione per l'analisi dei test e dei dettagli limitati delle user story.

Il testing esplorativo può essere applicato in vari punti del ciclo di vita dello sviluppo software Agile, tra cui:

- durante l'esecuzione dell'iterazione (per testare rapidamente nuove funzionalità o modifiche prima o parallelamente al testing guidato dagli script)
- nelle revisioni/demo delle iterazioni (per esplorare il prodotto in modo informale mentre si raccolgono feedback)
- dopo modifiche importanti (per individuare difetti di integrazione o regressione che i test basati su script potrebbero non rilevare)
- quando i criteri di accettazione sono vaghi o minimi (per chiarire le aspettative e individuare gli edge case), il testing esplorativo si basa in genere su test charter e viene eseguito in sessioni di test (Ghazi et al., 2017). Una sessione di test è un periodo di tempo prestabilito e ininterrotto (di solito 60-120 minuti). Queste sessioni di test aiutano a organizzare l'effort di test e forniscono una struttura in un approccio del test flessibile. Una sessione di test si focalizza su una specifica test charter ed è spesso documentata con schede di sessione di test, osservazioni e scoperte. La struttura tipica di una sessione di testing esplorativo è la seguente:
 - Configurazione pre-sessione. Viene definita la test charter, viene preparato l'ambiente di test e viene impostato il limite di tempo della sessione di test.
 - Fase di esecuzione dei test. Il tester apprende come funziona il prodotto, lo esplora e lo valuta. Il tester segue l'intento della test charter ma si adatta man mano che vengono scoperte nuove informazioni. Utilizza euristiche di test e mnemonici di test per simulare le condizioni di test. Poiché il testing esplorativo non è scriptato, una documentazione adeguata è essenziale per la riproducibilità, il reporting del test e il miglioramento. I metodi standard per prendere appunti includono schede di sessione, appunti in formato libero, screenshot, registrazioni dello schermo e mappe mentali. È possibile utilizzare anche strumenti di registrazione delle sessioni. Le informazioni documentate includono in genere la copertura (compresa la copertura dei rischi), le note di valutazione, la registrazione del comportamento effettivo del sistema e le anomalie rilevate.
 - Revisione post-sessione. Il tester esplorativo discute i risultati della sessione di test con gli stakeholder, tra cui:
 - il confronto tra la test charter e la realtà (il tester ha seguito la test charter, ha effettuato la scoperta di aree inaspettate e ha raggiunto gli obiettivi della sessione?);
 - difetti scoperti;

- domande sollevate (ad esempio, requisiti poco chiari o incertezze tecniche);
- passi successivi (sono necessarie nuove test charter o nuovi regression test?).

Quando si esegue il testing esplorativo, il tester non ha alcuna limitazione per quanto riguarda gli approcci del test, le tecniche di test o i tipi di test. Può utilizzare tecniche di test black-box e tecniche di test white-box. Tuttavia, data la natura del testing esplorativo, il testing basato sull'esperienza e le euristiche di test sono quelle più comunemente utilizzate (vedere la *Sezione 5.1.1*, la *Sezione 5.1.2* e la *Sezione 5.1.3*).

I tester esplorativi devono applicare gli oracoli del test in modo flessibile e contestuale, spesso facendo affidamento sulla loro esperienza, sulla conoscenza del prodotto e sulla collaborazione del team. Nel testing esplorativo, gli oracoli del test spesso:

- sono coerenti con le user story o i criteri di accettazione (vedere la *Sezione 5.1.4*)
- esprimono il buon senso o le aspettative degli utenti
- sono coerenti con funzionalità simili o versioni precedenti
- sono conformi a standard o linee guida (ad es. standard di accessibilità o di UI)

Le categorie di oracoli del test a supporto del testing esplorativo possono includere lo mnemonico di test FEW HICCUPPS (vedere la *Sezione 5.1.2*).

5.2 Testing Assistito

5.2.1 Mob testing

Il mob testing (noto anche come ensemble testing) è un approccio del test collaborativo in cui un piccolo gruppo (tipicamente 5-8 persone) testa contemporaneamente lo stesso sistema. Si basa sul concetto di swarming, in cui più membri del team convergono su un determinato problema di testing. Questa collaborazione in tempo reale combina competenze, prospettive e feedback immediato diversi per migliorare qualità, velocità e creatività del testing.

La configurazione del mob testing include uno schermo condiviso, un computer e una lavagna (in caso di lavoro a distanza l'ambiente può essere virtuale, piuttosto che fisico). Prevede i seguenti ruoli:

- Moderatore – coordinatore della sessione che supervisiona lo svolgimento della sessione, garantisce il rispetto dei ruoli, gestisce il tempo e favorisce una partecipazione equilibrata.
- Navigatore – il tester principale che prende le decisioni finali sulle azioni, incorporando le intuizioni o informazioni implicite e le indicazioni del mob.
- Driver – la persona che materialmente scrive e implementa le indicazioni concordate dal team. In genere non guida la strategia: in questo modo, prima di essere realizzate, le idee vengono elaborate anche dalla mente di un'altra persona.
- Mob – un gruppo di osservatori che monitora i progressi e contribuisce con suggerimenti quando necessario.

Il mob testing consiste di due parti: il mobbing e la retrospettiva. La sessione di mobbing inizia disponendo le persone in cerchio con il driver alla tastiera. Le persone ruotano ogni tanto (di solito ogni 4 minuti); il navigatore diventa il driver, il driver diventa la prima persona nel mob, e così via.

Il moderatore si siede in fondo e non ruota con il resto del mob. Se è necessario che intervenga, può mettere in pausa il mob e assumere qualsiasi ruolo sia necessario, tranne quello del driver. Il mob ruota frequentemente, costringendo tutti a prestare attenzione. La sessione di mobbing dura in genere 2 ore.

Al termine della sessione di mobbing, segue la retrospettiva per migliorare il processo in vista delle sessioni future. In genere, il moderatore raccoglie in silenzio le osservazioni di ciascun partecipante su foglietti adesivi. Il team raggruppa quindi gli elementi simili per individuare modelli e opportunità di miglioramento continuo.

5.2.2 Pair testing

Il pair testing è un approccio del test in cui due persone lavorano insieme su una singola postazione di lavoro (o in un ambiente virtuale in caso di lavoro a distanza o di un team distribuito) per testare lo stesso oggetto di test in tempo reale (ad esempio, tester + tester, tester + sviluppatore, tester + product analyst). In genere, nel pair testing, una persona utilizza il computer (il driver), mentre l'altra osserva, revisiona, prende appunti ed elabora strategie (il navigatore). Il pair testing trae ispirazione dal pair programming, una pratica resa popolare dall'Extreme Programming (XP).

Il pair testing favorisce la condivisione delle conoscenze incoraggiando l'apprendimento reciproco delle competenze del dominio e delle strategie di test, ed è particolarmente utile quando si inseriscono nuovi membri nel team. Lavorare in coppia aiuta a individuare i difetti più rapidamente grazie alla sinergia collaborativa che si crea. Ciò migliora anche la copertura combinando prospettive diverse. Il feedback in tempo reale aiuta a perfezionare le condizioni di test, il che porta a test case più creativi ed efficaci. Può anche migliorare la comunicazione riducendo i malintesi sui requisiti o sulle funzionalità. Il pair testing mantiene i tester coinvolti e riduce la probabilità di tralasciare passi del test. Il ciclo di feedback immediato può persino innescare correzioni istantanee dei difetti (che devono comunque essere segnalati in modo appropriato), specialmente nelle coppie sviluppatore-tester.

Il pair testing è particolarmente utile quando la collaborazione, l'apprendimento e il feedback rapido sono fondamentali. In particolare, è utile nel:

- testing esplorativo
- testing di funzionalità complesse o nuove
- testing interfunzionale (ad es. UI + API)
- mentoring di junior tester o formazione incrociata dei membri del team

Il pair testing non è privo di sfide. Richiede tempo ed effort, poiché due persone lavorano allo stesso compito, il che può sembrare meno efficiente rispetto al lavorare separatamente. Inoltre, i tester possono avere stili di lavoro, livelli di assertività o modi di pensare diversi. Questo può essere un vantaggio, ma può anche portare ad attriti, disagio e una riduzione della produttività. Una persona potrebbe dominare la sessione (ad esempio, facendo sempre il driver o prendendo le decisioni), mentre parlare costantemente, pensare ad alta voce e giustificare le decisioni può essere mentalmente estenuante.

Per superare queste sfide:

- il pair testing dovrebbe essere utilizzato in modo strategico, ad esempio solo per funzionalità complesse o ad alto rischio
- si dovrebbe praticare una comunicazione aperta e un ascolto attivo
- i ruoli dovrebbero essere chiaramente definiti (driver e navigatore) e alternati regolarmente per evitare schemi comportamentali fissi

- dovrebbero essere definiti obiettivi chiari prima di iniziare ogni sessione di pair testing

5.2.3 Vibe testing

Il vibe testing è un approccio del test emergente e informale, assistito dall'AI, che deve ancora essere completamente standardizzato. Si è sviluppato parallelamente alla crescita del codice generato dall'AI, comunemente noto come "vibe coding". Tipicamente, gli sviluppatori scrivono manualmente il codice, creano test case e seguono flussi di lavoro strutturati di Quality Control. Con il vibe coding, invece, gli sviluppatori descrivono il comportamento richiesto dall'applicazione in linguaggio naturale, tipicamente sotto forma di prompt per LLM (large language model), che poi generano il codice, spesso riducendo la necessità di una revisione umana dettagliata. Questo cambiamento nello sviluppo richiede un approccio del test, chiamato vibe testing, per verificare che il prodotto finale sia allineato all'intento dell'utente.

Il vibe testing enfatizza una validazione intent-first, focalizzandosi sul testare ciò che l'applicazione dovrebbe fare piuttosto che semplicemente su ciò che fa attualmente, invece di affidarsi a test script statici. Anziché specificare meticolosamente ogni test case in anticipo, i tester interagiscono con l'applicazione come farebbero utenti reali, affidandosi al testing esplorativo e a test case generati dall'AI. Il vibe testing accelera il processo di test, rende il testing più adattabile alle applicazioni basate sull'AI in rapida evoluzione, riduce la dipendenza dalla scrittura manuale dei test e si allinea strettamente allo stile di progettazione intuitivo e di alto livello del vibe coding. Tuttavia, presenta anche sfide. Poiché gli sviluppatori potrebbero non comprendere appieno o non effettuare la review del codice sottostante, esiste il rischio di difetti nascosti, vulnerabilità di sicurezza, allucinazioni dell'AI o comportamenti fraintesi. Il vibe testing mitiga questo rischio fungendo da salvaguardia, assicurando che l'applicazione fornisca la funzionalità prevista, piuttosto che solo ciò che l'AI ha interpretato.

Ad esempio, supponiamo che lo sviluppatore abbia utilizzato un LLM per generare codice con il prompt "Crea una pagina di login con campi e-mail e password. Gli utenti dovrebbero essere reindirizzati alla dashboard dopo aver effettuato correttamente l'accesso". Il tester utilizza prompt o l'esplorazione manuale per testare il comportamento del sistema dal punto di vista dell'utente. Esempi di prompt includono: "Testa l'accesso con un formato e-mail non valido e verifica che non si proceda alla dashboard" oppure "Dopo l'accesso, verifica che l'utente visualizzi la dashboard corretta in base al ruolo (ad es. amministratore o utente normale)". Il prompt originale dello sviluppatore non specificava le regole di validazione degli input, la gestione degli input errati o la logica della dashboard basata sui ruoli. Si tratta di aspettative comuni, ma a meno che non siano esplicitamente indicate, il LLM potrebbe ignorarle. Il vibe testing mira a eliminare queste potenziali lacune testando ciò che l'applicazione dovrebbe fare nel mondo reale, piuttosto che presumere che il LLM abbia capito tutto correttamente da un prompt vago.

5.3 Test Smell

Un test smell è un sintomo di un test mal progettato. Sebbene non indichi sempre un problema, suggerisce potenziali rischi e maggiori costi di manutenzione. Proprio come i code smell nello sviluppo del software dovrebbero essere affrontati per mantenere la qualità del codice, i test smell dovrebbero essere affrontati per mantenere una strategia di test efficace e affidabile. Questo Syllabus descrive solo i test smell nei test case manuali. Per i test case automatizzati, vedere (Aljedaani et al., 2021).

I seguenti test smell sono raggruppati in categorie, ognuna di esse con una descrizione del test smell e una soluzione raccomandata:

Smell di dipendenza e complessità

Test interdipendenti

- Smell: il successo di un test dipende dall'esecuzione preventiva di un altro test o dalla presenza di dati residui.
- Soluzione: rendere i test indipendenti, autonomi ed eseguibili in qualsiasi ordine.

Dipendenze nascoste

- Smell: il test case si basa su modifiche ai dati o alla configurazione non specificate nelle precondizioni.
- Soluzione: rendere le dipendenze esplicite e controllate.

Call on Me

- Smell: dipendenza da chiamate estese ad altri test case o dal passaggio di parametri tra di essi.
- Soluzione: evitare il nesting delle chiamate e il concatenamento dei test case; mantenere invece i test semplici e utilizzare nomi chiari e una documentazione adeguata.

ORacle

- Smell: utilizzo della logica decisionale nei passi del test, come l'offerta di azioni alternative e risultati attesi multipli.
- Soluzione: rendere i test case deterministici e privi di ramificazioni suddividendoli in test case separati, spostando la logica decisionale nelle precondizioni ed evitando l'uso di "o" nei risultati attesi.

Smell relativi ai risultati attesi

Risultati attesi non chiari

- Smell: le procedure di test elencano le azioni ma non definiscono chiaramente i criteri di superamento/fallimento.
- Soluzione: aggiungere i risultati attesi, utilizzando un formato strutturato che mantenga le aspettative visibili e tracciabili. Evitare un linguaggio soggettivo e stabilire la tracciabilità rispetto ai requisiti o ai criteri di accettazione.

One More Step

- Smell: i risultati attesi includono passi del test aggiuntivi, spesso utilizzando verbi d'azione come "controllare", "verificare" e "vedere".
- Soluzione: mantenere le azioni e i risultati attesi chiaramente separati dall'esecuzione dei test spostando la verifica in un proprio passo ed evitare di utilizzare verbi d'azione nei risultati attesi.

Smell relativi ai passi di test

Passi ambigui

- Smell: le istruzioni sono vaghe o soggette a interpretazione.
- Soluzione: utilizzare un linguaggio specifico e oggettivo con termini e dati concreti. Definire chiaramente gli input, evitare parole soggettive e fare riferimento a elementi o etichette specifici dell'UI.

Hotstepper

- Smell: i passi del test elencano ogni azione separatamente, anche quando l'azione non ha alcun effetto.
- Soluzione: unire i passi che non hanno un effetto specifico.

Passi sovraccarichi

- Smell: molte azioni sono incluse in un unico passo del test.
- Soluzione: utilizzare le precondizioni quando opportuno e seguire la regola "un'azione = un passo", assicurandosi che per ogni azione vengano aggiunti i risultati attesi corrispondenti.

Smell relativi al livello di dettaglio

Test case troppo dettagliati o troppo lunghi

- Smell: test case con molti passi del test che toccano più funzionalità o test case che includono il percorso di navigazione completo prima di ogni piccola azione.
- Soluzione: suddividere in test case più piccoli, modularizzare le parti riutilizzabili, rimuovere i dettagli di navigazione ripetuti e utilizzare efficacemente le precondizioni.

Test Case Monolitico (God Test Case)

- Smell: un singolo test case viene utilizzato per testare più scenari alternativi o addirittura l'intero sistema.
- Soluzione: progettare test case piccoli, focalizzati e indipendenti, organizzati in test suite.

Dati di test hardcoded

- Smell: ogni test case include esplicitamente tutti i propri dati di test.
- Soluzione: utilizzare descrizioni dei dati di alto livello dove appropriato e utilizzare parametri per separare i dati di test dal test case.

Smell dovuti a omissioni

Assunzioni ambientali

- Smell: il test case presuppone condizioni specifiche dell'ambiente di test senza documentarle.
- Soluzione: documentare esplicitamente le precondizioni dell'ambiente e i requisiti di configurazione.

Assenza di cleanup o teardown

- Smell: il test case non specifica come vengono gestiti i dati di test al completamento del test.
- Soluzione: definire procedure esplicite di cleanup o teardown.

Smell dovuti a eccessi

Invalidation Heaven

- Smell: ogni diverso tipo di input non valido viene gestito da un test case separato.
- Soluzione: combinare la gestione degli input non validi in un unico test case utilizzando i parametri.

Duplicazione tramite copia-incolla

- Smell: diversi test case differiscono solo per pochi valori dei dati di test.

- Soluzione: rifattorizzare in un unico test case parametrizzato e riutilizzabile per evitare la duplicazione, garantendo al contempo la tracciabilità rispetto ai requisiti per misurare la copertura.

Smell di formato e linguaggio

Formattazione non strutturata

- Smell: i passi del test non sono numerati e non presentano titoli.
- Soluzione: utilizzare modelli.

Refusi - Errori di scrittura

- Smell: i test case contengono errori ortografici e parole o frasi incomplete.
- Soluzione: eseguire un controllo ortografico.

Click-Push-Press

- Smell: la terminologia e la formulazione utilizzate nei test case sono incoerenti.
- Soluzione: utilizzare un glossario di espressioni standard.

6 Test Automation e Strumenti di Test - 30 minuti

Parole Chiave

sviluppo software Agile, test automation

Obiettivi di Apprendimento:

6.1 Test Automation nello Sviluppo Software Agile

CTAL-AT-6.1.1 (K2) Distinguere tra i diversi approcci di test automation applicabili allo sviluppo software Agile

6.2 Strumenti di Test nello Sviluppo Software Agile

CTAL-AT-6.2.1 (K2) Fornire esempi di strumenti di test utili nel testing Agile

Introduzione

La test automation può fornire un feedback rapido e affidabile in cicli di test brevi. La test automation integra, ma non sostituisce, il testing esplorativo e il testing manuale. Gli strumenti di test favoriscono la collaborazione, il tracciamento dei requisiti, il monitoraggio delle prestazioni e il reporting del test. Gli strumenti che si integrano agevolmente nello sviluppo software Agile migliorano la visibilità, accelerano il feedback e aiutano l'intero team a condividere la responsabilità della qualità.

6.1 Test Automation nello Sviluppo Software Agile

La decisione di automatizzare è guidata sia dal rischio sia dal valore. La test automation è particolarmente adatta ad attività ripetitive, deterministiche e di alto valore in termini di riduzione del rischio, specialmente quando i failure potrebbero impattare funzionalità business-critical. I test che non falliscono mai forniscono scarso valore e offrono benefici minimi se automatizzati. Allo stesso modo, i test il cui comportamento cambia frequentemente sono candidati poco adatti alla test automation, poiché i costi di manutenzione spesso superano i benefici.

È considerata una best practice decidere durante la pianificazione o il perfezionamento dell'iterazione quali user story automatizzare e in che misura. Ciò facilita una stima più accurata del tempo necessario per ottenere la piena accettazione della user story, con conseguente maggiore prevedibilità della consegna dell'iterazione.

Lo sviluppo software Agile adotta diversi approcci di test automation a seconda del contesto, delle competenze del team, della criticità del sistema e della frequenza di consegna. Un approccio del test comune consiste nell'automatizzare i test che forniscono un feedback rapido e una validazione ripetibile delle parti più preziose e stabili del sistema. Ciò include tipicamente test di componente, test delle API e un insieme di test end-to-end accuratamente definito, in conformità con la piramide di test (vedere: (ISTQB-CTFL, v4.0.1)). Il testing esplorativo e i test una tantum (ovvero i test eseguiti una sola volta per testare una specifica condizione di test in un determinato momento) vengono normalmente eseguiti manualmente perché si basano sull'intuizione umana, sono guidati da comportamenti emergenti o i costi di manutenzione sarebbero troppo elevati. Le stesse considerazioni si applicano spesso al testing di usabilità.

Anche la tempistica della test automation è una considerazione fondamentale. In un approccio test-first, i test vengono automatizzati precocemente, prima o durante la scrittura del codice. Ciò consente ai test di guidare lo sviluppo e fornire una verifica continua. Al contrario, l'automazione dei regression test è spesso un'attività che si evolve durante o dopo l'iterazione, a seconda di quando la funzionalità si stabilizza.

Il testing Agile integra la test automation durante tutta l'iterazione, a partire dalla verifica della build nella continuous integration, proseguendo con il testing esplorativo supportato da strumenti di test automation e concludendo con regression test suite automatizzate che confermano la stabilità del sistema prima della release.

Il successo della test automation nello sviluppo software Agile richiede una strategia a più livelli focalizzata sull'automazione di ciò che fornisce feedback coerente e prezioso, sulla selezione dei test giusti da automatizzare e sull'integrazione progressiva della test automation in tutto il ciclo di vita dello sviluppo software.

La piramide della test automation Agile, introdotta da Mike Cohn, promuove una strategia di testing bilanciata dando priorità a unit test veloci e di basso livello, supportandoli con un livello più ridotto di test

di servizio e di integrazione e mantenendo solo un insieme minimo di test della UI end-to-end, consentendo feedback rapido, costi ridotti e maggiore stabilità della CI/CD.

Altri Syllabi ISTQB forniscono conoscenze più approfondite sulla test automation (vedere: (ISTQB-TAE, v2.0), (ISTQB-TAS, v1.0)).

6.2 Strumenti di Test nello Sviluppo Software Agile

Gli strumenti di test utilizzati nel testing Agile abbracciano diverse categorie e supportano sia la natura iterativa che quella collaborativa dello sviluppo Agile. I team Agile traggono vantaggio da strumenti che facilitano la comunicazione, il feedback continuo e la test automation durante l'intero ciclo di vita dello sviluppo software. La selezione degli strumenti di test dovrebbe allinearsi alla strategia di test, ai livelli di test automation e agli obiettivi di feedback piuttosto che alla convenienza o alle tendenze.

Una varietà di strumenti supporta le attività di testing Agile e la collaborazione all'interno del team, ognuno con uno scopo distinto all'interno del processo di sviluppo e consegna del software:

Gli **strumenti di gestione delle attività e tracciamento** consentono al team di pianificare, prioritizzare e tracciare le user story, le attività di test e i difetti, favorendo la trasparenza e una comprensione condivisa dell'avanzamento del team e dei rischi. Questi strumenti sono essenziali per la gestione del backlog, dei difetti e delle test charter.

Gli **strumenti di comunicazione e condivisione delle informazioni** supportano la collaborazione in tempo reale, specialmente nei team distribuiti. Migliorano l'interazione durante le sessioni di perfezionamento, la pianificazione delle iterazioni, le test review e le retrospettive. Questi strumenti aiutano i tester a condividere informazioni, discutere i difetti e comunicare rapidamente il feedback dei clienti a tutto il team.

Gli **strumenti di progettazione dei test, di implementazione dei test e di esecuzione dei test** supportano la creazione e l'esecuzione sia di test automatizzati che di test manuali. Questi includono framework di test automation per lo unit testing, il testing delle API e la test automation della GUI. Consentono inoltre pratiche di specifica per esempi come ATDD e BDD, facilitando la collaborazione tra i membri del team sui criteri di accettazione.

Gli **strumenti di continuous integration** consentono una frequente integrazione del codice e un rapido rilevamento dei failure, mantenendo sempre il software pronto per il deployment. Questi strumenti sono essenziali nello sviluppo software Agile, dove cicli di feedback brevi supportano la consegna iterativa e aiutano a mitigare precocemente i rischi di regressione.

Gli **strumenti di configuration management** garantiscono la coerenza tra gli ambienti di test gestendo il codice, i test script e le modifiche all'infrastruttura. Questi strumenti sono spesso integrati nelle pipeline di CI/CD per supportare gli obiettivi DevOps e di continuous delivery.

Gli **strumenti di testing esplorativo** supportano il Session-Based Test Management, il tracciamento delle test charter, la registrazione delle schede della sessione di test e l'acquisizione di screenshot. Questi strumenti aiutano a catturare condizioni di test e risultati dei test con un sovraccarico di documentazione minimo.

Gli **strumenti di monitoraggio e analisi** forniscono approfondimenti a livello di produzione tracciando i modelli di utilizzo, i tassi di difetti e le metriche di prestazione. Sono particolarmente utili nei team Agile che praticano il monitoraggio come testing e in quelli che utilizzano il feedback degli utenti reali per perfezionare il comportamento del prodotto dopo il deployment.

Gli **strumenti di AI** possono generare automaticamente test case da modifiche al codice, requisiti o modelli di comportamento degli utenti; prevedere failure della build utilizzando modifiche al codice e dati storici; analizzare user story o requisiti per rilevare ambiguità, criteri di accettazione mancanti o terminologia incoerente; e creare dati di test realistici che coprano gli edge case.

7 Riferimenti

Standard

IEEE 1044 – Classification for Software Anomalies, 2009. Standard. Institute of Electrical and Electronics Engineers.

ISO 29119-6 Software and systems engineering – Software testing: Part 6: Guidelines for the use of ISO/IEC/IEEE 29119 (all parts) in agile projects, 2021. Technical report. International Organization for Standardization.

Documenti ISTQB®

ISTQB-ATLAS, ISTQB®, 2023. *Certified Tester Agile Test Leadership at Scale: Syllabus*. Versione 1.0.

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

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

ISTQB-TAS, ISTQB®, 2024. *Certified Tester Test Automation Strategy: Syllabus*. Versione 1.0.

Riferimenti al Glossario

Riferimenti per la terminologia utilizzata in questo documento:

- Agile Alliance: <https://agilealliance.org/agile101/agile-glossary>
- IEEE/Pascal for software engineering: https://pascal.computer.org/sev_display/index.action
- IREB-CPRE for Requirements Engineering: <https://cpre.ireb.org/en/downloads-and-resources/glossary>
- ISTQB® Glossary: <https://glossary.istqb.org/>

Libri

ADZIC, Gojko, 2011. *Specification by Example: How Successful Teams Deliver the Right Software*. Manning.

ANDERSON, D.J.; REINERSTEN, D.G., 2010. *Kanban: Successful Evolutionary Change for Your Technology Business*. Blue Hole Press.

ANDERSON, L.W.; KRATHWOHL, D.R., 2001. *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*. Longman. isbn 9780801319037.

BECK, K.; ANDRES, C., 2004. *Extreme Programming Explained: Embrace Change, 2nd Ed*. Addison-Wesley.

BEIZER, Boris, 1990. *Software Testing Techniques*. Van Nostrand Reinhold: Boston MA.

GREAVES, K.; LAING, S., 2019. *Growing Agile: A Coach's Guide to Agile Testing*. Leanpub.

GREGORY, J.; CRISPIN, L., 2019. *Agile Testing Condensed: A Brief Introduction*. Leanpub.

WHITTAKER, J.A., 2009. *Exploratory Software Testing: Tips, Tricks, Tours, and Techniques to Guide Test Design*. Addison-Wesley Professional.

Articoli

ALJEDAANI, Wajdi; PERUMA, Anthony; ALJOHANI, Ahmed; ALOTAIBI, Mazen; MKAOUER, Mohamed Wiem; OUNI, Ali; NEWMAN, Christian D.; GHALLAB, Abdullatif; LUDI, Stephanie, 2021. Test Smell Detection Tools: A Systematic Mapping Study, pp. 170–180. Disponibile su doi: 10.1145/3463274.3463335.

GHAZI, Ahmad Nauman; GARIGAPATI, Ratna Pranathi; PETERSEN, Kai, 2017. Checklists to Support Test Charter Design in Exploratory Testing, pp. 251–258. isbn 978-3-319-57633-6.

NIELSEN, Jakob, 1994. Enhancing the explanatory power of usability heuristics. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 152–158.

WALKER, R.; CENYDD, L. ap; POP, S.; MILES, H. C.; HUGHES, C. J.; TEAHAN, W. J.; ROBERTS, J. C., 2013. Storyboarding for visual analytics. *Information Visualization*. Vol. 14, pp. 27–50.

Pagine Web

Agile Alliance, [n.d.]. Disponibilità anche su: <https://agilealliance.org/agile101/agile-glossary>.

BECK, K.; BEEDLE, M.; BENNEKUM, A. van; COCKBURN, A.; CUNNINGHAM, W.; FOWLER, M.; GRENNING, J.; HIGHSMITH, J.; HUNT, A.; JEFFRIES, R.; KERN, J.; MARICK, B.; MARTIN, R.; MELLOR, S.; SCHWABER, K.; SUTHERLAND, J.; THOMAS, D., 2001. *Agile Manifesto* [online]. [consultato il 30/08/2025]. Disponibilità su: <https://agilemanifesto.org/>.

BOLTON, M., 2012. *FEW HICCUPS* [online]. [consultato il 27/07/2025]. Disponibilità su: <https://developsense.com/blog/2012/07/few-hiccups>.

BOLTON, M., 2014. *How Models Change* [online]. [consultato il 27/07/2025]. Disponibilità su: <https://developsense.com/blog/2014/07/how-models-change>.

GAREEV, A., 2019. *The TERMS for Test Automation* [online]. [consultato il 27/07/2025]. Disponibile su: <https://www.automation-beyond.com/2019/04/10/the-terms-for-test-automation>.

GREGORY, J., 2021. *Testing from a holistic point of view* [online]. [consultato il 05/09/2025]. Disponibile su: <https://janetgregory.ca/testing-from-a-holistic-point-of-view/>.

IEEE/Pascal for software engineering, [n.d.]. Disponibilità anche su: https://pascal.computer.org/sev_display/index.action.

IREB-CPRE for Requirements Engineering, [n.d.]. Disponibilità anche su: <https://cpre.ireb.org/en/downloads-and-resources/glossary>.

JOHNSON, K., 2012. *Software Testing Heuristics and Mnemonics* [online]. [consultato il 27/07/2025]. Disponibilità su: <https://www.slideshare.net/slideshow/kn-johnson-2012-heuristics-mnemonics/15019733>.

KOHL, J., 2010. *Test Mobile Applications with I SLICED UP FUN!* [online]. [consultato il 27/07/2025]. Disponibilità su: <https://www.kohl.ca/articles/ISLICEDUPFUN.pdf>.

OWASP, 2025. *Top Ten Web Application Security Risks* [online]. 27/07/2025. [consultato il 27/07/2025]. Disponibile su: <https://owasp.org/www-project-top-ten>.

PROSCI, 2026. *The Prosci ADKAR Model* [online]. 04/01/2026. [consultato il 04/01/2026]. Disponibilità su: <https://www.prosci.com/methodology/adkar>.

SUTHERLAND, J.; SCHWABER, K., 2020. *Scrum Guide* [online]. [consultato il 30/08/2025]. Disponibilità su: <https://scrumguides.org/index.html>.

WYNNE, M., 2015. *Introducing Example Mapping* [online]. [consultato il 07-10-2025]. Disponibilità su:
<https://cucumber.io/blog/bdd/example-mapping-introduction>.

I riferimenti precedenti rimandano a informazioni disponibili su Internet e altrove. Sebbene tali riferimenti siano stati verificati al momento della pubblicazione di questo Syllabus, ISTQB® non può essere ritenuta responsabile qualora la disponibilità dei riferimenti non fosse più garantita.

8 Ulteriori Letture

GREGORY, J.; CRISPIN, L., 2024. *Holistic Testing: Weave Quality into Your Product*. Leanpub.

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

ISO 29119-1 *Software and systems engineering – Software testing: Part 1: General concepts*, 2022. Standard. International Organization for Standardization.

ISO/IEC 20246: *Software and systems engineering – Work product reviews*, 2017. Standard. International Organization for Standardization.

ISO/IEC 25010: *Systems and software Quality Requirements and Evaluation (SQuaRE) – Product quality model*, 2023. Standard. International Organization for Standardization.

ISO/IEC 25019: *Systems and software Quality Requirements and Evaluation (SQuaRE) – Quality-in-use model*, 2023. Standard. International Organization for Standardization.

ISO/IEC/IEEE 29119-3: *Software Testing, Part 3: Test documentation*, 2021. Standard. International Organization for Standardization.

ISO/IEC/IEEE 29119-4: *Software testing, Part 4: Test techniques*, 2021. Standard. International Organization for Standardization.

ISO/IEC/IEEE 29119-5: *Software testing, Part 5: Keyword-driven testing*, 2016. Standard. International Organization for Standardization.

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

ISTQB-ITP, ISTQB®, 2011. *Certified Tester Expert Level Improving the Test Process: Syllabus*. Versione 1.0.

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

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

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

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

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

9 Appendice A – Elenco delle Abbreviazioni

Abbreviazione	Definizione
AI	Artificial Intelligence
API	Application Programming Interface
ATDD	Acceptance Test-Driven Development
BDD	Behavior-Driven Development
BPMN	Business Process Model and Notation
CI/CD	Continuous Integration/Continuous Delivery
GQM	Goal-Question-Metric
LLM	Large Language Model
MTTR	Mean Time to Recovery
OWASP	Open Web Application Security Project
TDD	Test-Driven Development
UI	User Interface
XP	eXtreme Programming

10 Appendice B – Termini Specifici del Dominio

Nome del termine	Definizione
bias	Una deviazione sistematica dall'obiettività che può influenzare gli obiettivi del test, le attività di test o i risultati dei test.
canary release	Una strategia di deployment in cui una nuova versione di un sistema viene rilasciata a un sottoinsieme limitato di utenti per valutarla prima di un'implementazione più ampia a tutti gli utenti.
community of practice	Un gruppo di professionisti che condividono un interesse, una passione o una sfida comune, si incontrano regolarmente per condividere conoscenze, imparare dalle reciproche esperienze e migliorare le competenze per risolvere problemi e innovare nel proprio dominio specifico.
dark launch	Una tecnica in cui funzionalità nuove o modificate vengono implementate nell'ambiente di produzione senza essere visibili o accessibili agli utenti finali, al fine di validare il comportamento, le prestazioni o il rischio in condizioni reali prima della release completa.
Definition of Done	Una descrizione formale dello stato dell'incremento quando soddisfa i requisiti di qualità richiesti per il prodotto.
epic	Una descrizione di un'esigenza degli stakeholder che è tipicamente più ampia di quanto possa essere implementato in una singola iterazione.
example mapping	Una tecnica collaborativa nello sviluppo software Agile che utilizza regole, esempi e domande per chiarire i requisiti e supportare la progettazione dei test.
feature toggle	Un meccanismo per modificare il comportamento del sistema senza cambiare il codice.
FEW HICCUPPS	Un'euristica di test per Familiarity, Explainability, World, History, Image, Comparable product, Claims, Users' desires, Product, Purpose, Statutes, a supporto dell'identificazione degli oracoli del test.
iterazione di hardening	Un'iterazione, tipicamente condotta alla fine di una serie di iterazioni di sviluppo, in cui l'obiettivo principale è stabilizzare il prodotto risolvendo i difetti in sospeso, migliorando le prestazioni, completando le attività di integrazione e di testing di sistema e preparando il prodotto per la release.
I SLICED UP FUN	Un'euristica di test per Inputs, Store, Location, Interactions/Interruptions, Communication, Ergonomics, Data, Usability, Platform, Function, User scenarios, Network.
iterazione	Un ciclo di durata fissa e a tempo determinato in cui viene eseguita una serie di attività pianificate per produrre una versione potenzialmente rilasciabile di un prodotto.
metrica	Una scala di misurazione e un metodo utilizzato per la misurazione.
RCRCRC	Un'euristica di test per Recent, Core, Risky, Configuration, Repaired, Chronic.
risk-storming	Una tecnica di brainstorming strutturato in cui gli stakeholder identificano in modo collaborativo i potenziali rischi al fine di supportare il testing basato sul rischio.
SFDIPOT	Un'euristica di test per Structure, Function, Data, Interfaces, Platform, Operations, Time.

specifica per esempi	Una tecnica di sviluppo in cui la specifica è definita per esempi.
storyboarding	Una rappresentazione visiva di una sequenza di interazioni dell'utente o di comportamenti del sistema, utilizzata per chiarire i requisiti, supportare la progettazione dei test e comunicare gli scenari in modo efficace.
conversazione strutturata	Una discussione facilitata e mirata che segue una struttura definita o una serie di spunti per garantire una comprensione condivisa, concentrarsi sugli obiettivi e uno scambio efficace di informazioni tra gli stakeholder.
TERMS	Un'euristica del test per Tools and Technology, Execution, Requirements and Risks, Maintenance, Security.
scheda di sessione di test	Un documento utilizzato nel testing esplorativo per registrare l'esecuzione e i risultati di una sessione di test a tempo determinato, comprese le note sulle attività di test eseguite, le osservazioni, la copertura di test raggiunta, i difetti rilevati e gli elementi di follow-up.
user journey	Una rappresentazione della sequenza di passaggi che un utente esegue per raggiungere un obiettivo specifico quando interagisce con un sistema, comprese l'esperienza, le percezioni e le risposte emotive dell'utente ad ogni passaggio.
user story	Una breve narrazione che descrive un'esigenza dal punto di vista dell'utente insieme al beneficio atteso quando tale esigenza viene soddisfatta.
user story slicing	La pratica di suddividere una user story ampia o complessa in user story più piccole, gestibili e testabili.

11 Appendice C – Obiettivi di Apprendimento/Livello Cognitivo di Conoscenza

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

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

Livello 1: Ricordare (K1)

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

Verbi di azione: richiamare, riconoscere.

Esempi
Richiamare i concetti della piramide di test.
Riconoscere gli obiettivi tipici del testing.

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

Esempi	Note
Classificare gli strumenti di testing in base al loro scopo e alle attività di testing che supportano.	
Confrontare i diversi livelli di test.	Può essere utilizzato per cercare somiglianze, differenze o entrambe le cose.
Distinguere il testing dal debugging.	Cerca le differenze tra i concetti.
Distinguere tra rischi di progetto e rischi di prodotto.	Consente di classificare separatamente due (o più) concetti.
Spiegare l'impatto del contesto sul processo di test.	
Fornire esempi del perché il testing è necessario.	
Ricavare la root cause dei difetti da un determinato profilo di failure.	
Riassumere le attività del processo di review del prodotto di lavoro.	

Livello 3: Applicare (K3)

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

Verbi di azione: applicare, implementare, preparare, utilizzare.

Esempi	Note
Applicare l'analisi ai valori limite per ricavare i test case da determinati requisiti.	Dovrebbe riferirsi a una procedura / tecnica / processo ecc.
Implementare metodi di raccolta delle metriche per supportare i requisiti tecnici e di management.	
Preparare test di installabilità per le applicazioni mobile.	
Utilizzare la tracciabilità per monitorare l'avanzamento del test, in modo da poter verificare la completezza e la consistenza con gli obiettivi del test, la strategia di test e il test plan.	Potrebbe essere utilizzato in un LO che richiede che il candidato sia in grado di utilizzare una tecnica o una procedura. Simile ad "applicare".

Livello 4: Analizzare (K4)

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

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

Esempi	Note
Analizzare una determinata situazione di progetto per determinare quali tecniche di testing black-box o basate sull'esperienza dovrebbero essere applicate per raggiungere obiettivi specifici.	Esaminabile solo in combinazione con un obiettivo misurabile dell'analisi. Dovrebbe essere della forma "Analizzare xxxx per xxxx" (o similare).
Prioritizzare i test case in una determinata test suite per l'esecuzione in base ai rischi di prodotto correlati.	
Selezionare i livelli di test e i tipi di test appropriati per verificare un determinato insieme di requisiti.	Necessario quando la selezione richiede un'analisi.

Livello 5: Valutare (K5)

Il candidato può formulare giudizi sulla base di criteri e standard. Individua incongruenze o errori all'interno di un processo o di un prodotto, determina se un processo o un prodotto presenta coerenza interna e individua l'efficacia di una procedura mentre viene implementata (ad esempio, determinare se le conclusioni di uno scienziato derivano dai dati osservati).

Verbi di azione: valutare, criticare, giudicare, raccomandare.

Esempi	Note
Valutare un'organizzazione di test utilizzando TPI Next o TMMi.	
Criticare l'adeguatezza delle attività di test in un'organizzazione per il determinato contesto.	

Valutare un'organizzazione per determinare le opzioni per il corretto posizionamento del team di test.	Dovrebbe limitarsi alla valutazione, senza includere la soluzione risultante (che sarebbe K6)
Raccomandare misure per ottenere l'accettazione dei cambiamenti da parte delle persone coinvolte.	

Livello 6: Creare (K6)

Il candidato mette insieme gli elementi per formare un insieme coerente o funzionale. Un'applicazione tipica consiste nel riorganizzare gli elementi in un nuovo modello o struttura, ideare una procedura per portare a termine un compito o inventare un prodotto (ad esempio, costruire habitat per uno scopo specifico).

Verbi di azione: creare, progettare, sviluppare, pianificare.

Esempi	Note
Creare un piano di miglioramento dei test tenendo conto delle questioni relative al change management con passaggi e azioni appropriati.	
Progettare una struttura organizzativa per un determinato ambito di un programma di test process improvement	
Sviluppare un processo di defect management per un'organizzazione di test, inclusi il flusso di lavoro del defect report e la comunicazione	
Pianificare ed eseguire interviste di valutazione utilizzando un particolare modello basato sul processo o sul contenuto.	

I livelli cognitivi degli obiettivi di apprendimento si basano su (L. Anderson et al., 2001).

12 Appendice D – Matrice di Tracciabilità dei Business Outcome agli Obiettivi di Apprendimento

Questo capitolo elenca la tracciabilità tra i Business Outcome e gli Obiettivi di Apprendimento dell'Advanced Level – Agile Tester.

Business Outcome: Advanced Level Agile Tester			BO1	BO2	BO3	BO4	BO5	BO6	BO7	BO8
CTAL-AT_BO1	Collaborare in team interfunzionali, avendo familiarità con i principi e le pratiche di base dello sviluppo software Agile		6							
CTAL-AT_BO2	Adattare l'esperienza e le conoscenze di testing esistenti ai valori e ai principi Agile			8						
CTAL-AT_BO3	Supportare il team Agile nella pianificazione dei test				6					
CTAL-AT_BO4	Applicare approcci di sviluppo software Agile e tecniche di test rilevanti per garantire che i test forniscano una copertura adeguata					10				
CTAL-AT_BO5	Assistere gli stakeholder di business nella definizione di user story, scenari, requisiti e criteri di accettazione comprensibili e testabili, secondo necessità						6			

Business Outcome: Advanced Level Agile Tester			BO1	BO2	BO3	BO4	BO5	BO6	BO7	BO8
CTAL-AT_BO6	Creare e implementare vari approcci del test per lo sviluppo software Agile							10		
CTAL-AT_BO7	Supportare e contribuire alla test automation in un progetto di sviluppo software Agile								2	
CTAL-AT_BO8	Lavorare con gli altri membri del team e condividere informazioni con loro utilizzando stili e canali di comunicazione efficaci									9
LO unico	Obiettivo di Apprendimento	Livello K								
1	Sfide relative alla Strategia di Test e all'Approccio del Test - 60 minuti									
1.1	Tipi di Test									
CTAL-AT-1.1.1	Confrontare i tipi di test da eseguire durante e dopo un'iterazione	K2		X	X					
1.2	Testing End-to-End	K2								
CTAL-AT-1.2.1	Spiegare quando dovrebbe essere eseguito il testing end-to-end	K2		X	X					
1.3	Testing Formale e Testing Olistico									

Business Outcome: Advanced Level Agile Tester			BO1	BO2	BO3	BO4	BO5	BO6	BO7	BO8
CTAL-AT-1.3.1	Confrontare i vantaggi e gli svantaggi del testing formale e del testing olistico	K2		X						
1.4	Approcci del Regression Testing									
CTAL-AT-1.4.1	Distinguere tra i diversi approcci del regression testing	K2		X	X			X		
2	Persone e Team - 60 minuti									
2.1	Whole Team Approach									
CTAL-AT-2.1.1	Confrontare generalizzazione e specializzazione all'interno di un team	K2	X							X
CTAL-AT-2.1.2	Fornire esempi di come motivare i rappresentanti del business a svolgere attività di test	K2	X							X
CTAL-AT-2.1.3	Riassumere in che modo il whole team approach possa aiutare il team di sviluppo	K2	X							X
2.2	Tissue Tester									
CTAL-AT-2.2.1	Spiegare come e quando utilizzare i tissue tester	K2				X				

Business Outcome: Advanced Level Agile Tester			BO1	BO2	BO3	BO4	BO5	BO6	BO7	BO8
3	Test Management e Test Process Improvement - 210 minuti									
3.1	Pianificazione dei Test									
CTAL-AT-3.1.1	Riassumere come eseguire la pianificazione dei test nello sviluppo software Agile	K2	X	X	X					
CTAL-AT-3.1.2	Delineare una strategia di test del progetto utilizzando i quadranti del testing	K4		X				X		
3.2	Monitoraggio dei Test e Controllo dei Test									
CTAL-AT-3.2.1	Spiegare come eseguire il monitoraggio dei test e il controllo dei test nello sviluppo software Agile	K3			X					
3.3	Reporting del Test									
CTAL-AT-3.3.1	Confrontare i diversi tipi di copertura che possono essere utilizzati per il reporting del test nello sviluppo software Agile	K2		X		X				
3.4	Test Process Improvement									

Business Outcome: Advanced Level Agile Tester			BO1	BO2	BO3	BO4	BO5	BO6	BO7	BO8
CTAL-AT-3.4.1	Selezionare misure appropriate di Test Process Improvement basate su metriche nello sviluppo software Agile	K4	X	X						X
CTAL-AT-3.4.2	Spiegare come eseguire il Test Process Improvement nello sviluppo software Agile	K2	X		X					X
4	Shift-Left - 135 minuti									
4.1	Utilizzare lo Shift-Left per Migliorare la Qualità della Base di Test									
CTAL-AT-4.1.1	Fornire esempi di come il testware possa essere utilizzato come una forma di requisiti	K2					X			X
CTAL-AT-4.1.2	Spiegare come lo storyboarding e il testboarding possano essere utilizzati per aumentare la qualità della base di test	K2					X			X
CTAL-AT-4.1.3	Spiegare come l'example mapping possa essere utilizzato per aumentare la qualità della base di test	K2					X			X
CTAL-AT-4.1.4	Fornire esempi di come i bias possano influire negativamente sulla qualità del prodotto	K2					X			

Business Outcome: Advanced Level Agile Tester			BO1	BO2	BO3	BO4	BO5	BO6	BO7	BO8
CTAL-AT-4.1.5	Applicare lo user story slicing per ottenere user story testabili	K3					X			
4.2	Shift-Left e Requirements Engineering									
CTAL-AT-4.2.1	Spiegare in che modo il requirements engineering supporta lo shift-left	K2					X			
5	Approcci e Tecniche di Test Agile - 285 minuti									
5.1	Testing Esplorativo									
CTAL-AT-5.1.1	Spiegare le euristiche di test	K2				X		X		
CTAL-AT-5.1.2	Fornire esempi di mnemonici di test relativi al testing nello sviluppo software Agile	K2				X		X		
CTAL-AT-5.1.3	Spiegare i tour di test	K2				X		X		
CTAL-AT-5.1.4	Analizzare le user story e le epic per creare le test charter	K4								X
CTAL-AT-5.1.5	Applicare il testing esplorativo a supporto del testing nello sviluppo software Agile	K3				X		X		

Business Outcome: Advanced Level Agile Tester			BO1	BO2	BO3	BO4	BO5	BO6	BO7	BO8
5.2	Testing Assistito									
CTAL-AT-5.2.1	Spiegare il mob testing	K2				X		X		
CTAL-AT-5.2.2	Spiegare il pair testing	K2				X		X		
CTAL-AT-5.2.3	Spiegare il vibe testing	K2				X		X		
5.3	Test Smell									
CTAL-AT-5.3.1	Utilizzare i test smell per valutare la qualità dei test case	K3				X		X		
6	Test Automation e Strumenti di Test - 30 minuti									
6.1	Test Automation nello Sviluppo Software Agile									
CTAL-AT-6.1.1	Distinguere tra i diversi approcci di test automation applicabili allo sviluppo software Agile	K2							X	
6.2	Strumenti di Test nello Sviluppo Software Agile									

Business Outcome: Advanced Level Agile Tester			BO1	BO2	BO3	BO4	BO5	BO6	BO7	BO8
CTAL-AT-6.2.1	Fornire esempi di strumenti di test utili nel testing Agile	K2							X	

13 Appendice E – Release Note

Questa è una versione principale del Syllabus, pertanto non vengono fornite modifiche dettagliate rispetto alla versione precedente. Tuttavia, poiché la differenza tra la v1.0 e la v2.0 è significativa, di seguito vengono descritte la natura generale e la motivazione delle modifiche.

Poiché Agile è ormai un metodo standard di sviluppo software, le nozioni di base di Agile non sono più descritte negli obiettivi di apprendimento. Queste nozioni di base sono invece trattate nel Capitolo 1: Introduzione all'Agile. Parte del contenuto del Syllabus Agile Tester v1.0 è stata spostata nel Syllabus Foundation Level v4.0. Pertanto, tali contenuti non sono trattati nell'attuale Syllabus Agile Tester v2.0. Questo vale per l'approccio test-first, la piramide di test, i quadranti del testing Agile, il testing indipendente, la creazione collaborativa di user story, le retrospettive e la stima dell'effort di test. Alcuni di questi contenuti sono approfonditi in questo Syllabus (ad esempio i quadranti del testing Agile al livello K4).

Sono stati introdotti nuovi argomenti (come mnemonici, euristiche, tour di test, bias, metriche di copertura per il reporting, testing olistico, tissue tester, mob testing, vibe testing, storyboarding e example mapping) e alcuni dei contenuti discussi nel Syllabus del Foundation Level 4.0 sono stati ampliati (ad esempio, il testing esplorativo e il whole team approach).

14 Marchi Registrati

ISTQB® è un marchio registrato di International Software Testing Qualifications Board. TMMi® è un marchio registrato di TMMi Foundation. TPI-Next® è un marchio registrato di Sogeti, Paesi Bassi.

15 Indice

Acceptance Test-Driven Development; 13;
17; 21; 23; 24; 35; 36; 38; 62

ambiente di test; 30; 31; 32; 43; 45; 47; 56

analisi ai valori limite; 30

analisi dei modelli di failure; 30

approccio del test; 16; 27; 28; 33; 47; 48; 49;
50; 55

approccio test-first; 33; 55

autovalutazione; 33

Behavior-Driven Development; 21; 23; 35; 36;
38; 62

bias; 25; 35; 36; 38; 39; 63

bug bash; 13; 20; 23

burn-down chart; 30

canary release; 20; 26; 33; 63

checklist; 33; 36; 37; 43; 44

cicli di feedback; 20; 23; 30; 31; 32; 33; 36;
38; 49; 56

continuous delivery; 15; 20; 22; 29; 31; 36;
39; 56

continuous integration; 15; 28; 33; 39; 55; 56

controllo dei test; 26; 27; 30

controllo del rischio; 18

copertura; 19; 20; 23; 27; 28; 29; 30; 31; 32;
34; 36; 39; 44; 47; 49; 53; 64

copertura del codice; 17; 24; 31; 32

copertura della test automation; 30; 43

creazione collaborativa di user story; 15

criteri di accettazione; 16; 17; 20; 22; 23; 24;
28; 29; 30; 31; 36; 37; 38; 39; 40; 41; 45;
46; 47; 48; 51; 56; 57

dashboard automatizzate; 30

dati di test; 32; 46; 52; 57

debito di test; 32

Definition of Done; 16; 26; 28; 29; 63

demo; 23; 47

DevOps; 20; 22; 25; 31; 36; 56

esempi; 15; 23; 24; 31; 36; 37; 38; 39; 63; 64

euristica; 42; 43

example mapping; 35; 36; 37; 38; 39; 63

Extreme Programming; 15; 49; 62

feature toggle; 13; 18; 20; 26; 30; 33; 63

feedback continuo; 31; 56

flaky test; 32

flow diagram cumulativi; 30

generalizzazione; 22

Goal-Question-Metric; 26; 33; 62

identificazione del rischio; 37

Kanban; 15

large language model; 50; 62

livello di test; 17; 27; 28; 43; 45

Manifesto Agile; 14

Manifesto del Testing Agile; 15

mappa mentale; 37; 47

mean time to failure; 32

mean time to recovery; 33; 62

mean time to repair; 32

metrica; 26; 27; 30; 31; 32; 33; 34; 63

mnemonico per il test; 42; 44; 47

mob testing; 42; 48

monitoraggio; 18; 20; 24; 29; 31; 33; 36; 55;
56

monitoraggio dei test; 26; 27; 30

oracolo del test; 44; 46; 48; 63

osservabilità; 18; 20; 29; 30; 31; 32

pair testing; 33; 39; 42; 49
pensiero critico; 33; 34; 44
performance testing; 29
pianificazione; 15; 23; 28
pianificazione della release; 28
pianificazione dell'iterazione; 28; 37; 55; 56
piramide di test; 18; 32; 55
product backlog; 15; 28
prototipi; 29
quadranti del testing; 26; 27; 29
qualità; 14; 15; 16; 17; 19; 22; 23; 27; 30; 35; 37; 48
regression testing; 13; 17; 18; 20; 28; 32; 43; 44
requisito; 14; 15; 18; 19; 23; 27; 31; 36; 37; 38; 40; 47; 48; 49; 51; 55; 57
retrospettiva; 15; 23; 30; 32; 33; 34; 49; 56
revisione delle iterazioni; 30; 47
rischio; 17; 19; 20; 22; 24; 27; 30; 31; 33; 38; 43; 50; 55
rischio di qualità; 16; 23; 28
rischio di regressione; 29; 56
root cause analysis; 32; 33; 39
Scrum; 15
Session-Based Test Management; 31; 32; 34; 56
shift-left; 15; 35; 36; 38; 40
simulazioni; 29; 30
smoke test; 20
smoke testing; 29
specializzazione; 22
specifica per esempi; 23; 24; 35; 36; 39; 56; 64
stakeholder di business; 14; 16; 36
stand-up quotidiani; 30
state diagram; 37
storyboarding; 35; 36; 37; 64
strategia di test; 18; 26; 27; 29; 30; 32; 34; 43; 49; 50; 56
tasso di difetti sfuggiti; 33
team interfunzionale; 29
tecniche di test basate sull'esperienza; 29
telemetria; 30
tempi di ciclo; 32
test automation; 18; 19; 22; 28; 29; 32; 34; 39; 44; 54; 55; 56
test charter; 31; 36; 38; 42; 44; 45; 56
test double; 30
test process improvement; 26; 27; 32; 33
test smell; 42; 50
testabilità; 23; 29; 39
testboarding; 35; 36; 37
Test-Driven Development; 13; 15; 17; 62
testing A/B; 18; 30
testing black-box; 17
testing collaborativo; 15
testing della tabella delle decisioni; 30
testing delle API; 29; 56
testing delle transizioni di stato; 30
testing di affidabilità; 29
testing di compatibilità; 29
testing di componente; 18; 29; 32
testing di sicurezza; 29
testing di usabilità; 22; 23; 25; 28; 29; 31; 55
testing end-to-end; 13; 17; 18; 39
testing esplorativo; 15; 17; 20; 22; 24; 25; 28; 29; 30; 31; 32; 33; 36; 38; 39; 42; 44; 45; 47; 48; 49; 50; 55; 56; 64
testing formale; 13; 18; 19
testing funzionale; 17; 28; 29; 30
testing non-funzionale; 17; 29; 30

testing olistico; 13; 19	user story slicing; 35; 36; 39; 40; 64
testing white-box; 17	valutazione del rischio; 20; 36
tipo di test; 13; 16; 27; 28; 43; 48	verifica della build; 55
tissue tester; 21; 22; 24; 39	vibe coding; 50
tour; 42; 44	vibe testing; 42; 50
user acceptance testing; 29; 30	walkthrough da parte dell'intero team; 20
user journey; 13; 17; 18; 35; 37; 64	whole team approach; 15; 22; 24; 32; 38