

Certified Tester Quality in DevOps (CT-QDO) Syllabus

v1.0

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 Szilárd Széll (product owner), Kari Kakkonen, Rik Marselis, Katja Obring e Yaron Tsubery, e gli autori dell'esame di esempio Tal Pe'er, Daniel Pol'an, Tomas Tumasonis, Aneta Derková.

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 titolari 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 titolari 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
v1.0	17/04/2026	Versione di Rilascio

Cronologia delle versioni della traduzione italiana del syllabus v1.0

Versione	Data	Osservazioni
Ver. A	16/09/2026	Prima versione pubblicata della traduzione italiana

Indice dei Contenuti

Avviso di Copyright	2
Cronologia delle Revisioni	3
Indice dei Contenuti	4
Ringraziamenti	6
0 Introduzione	7
0.1 Scopo di questo Syllabus	7
0.2 Il Certified Tester Quality in DevOps	7
0.3 Percorso di Carriera per i Tester	7
0.4 Business Outcome	8
0.5 Obiettivi di Apprendimento, Obiettivi Pratici e Livello Cognitivo di Conoscenza	8
0.6 L'Esame per la Certificazione Certified Tester Quality in DevOps	9
0.7 Accreditamento	10
0.8 Gestione degli Standard	10
0.9 Livello di Dettaglio	10
0.10 Com'è Organizzato questo Syllabus	10
1 Fondamenti di DevOps - 140 minuti	12
1.1 Quality Assurance (QA) in DevOps	13
1.1.1 Concetti Chiave di DevOps	13
1.1.2 Abbattere il Wall of Confusion	14
1.1.3 Le Metriche di Performance DORA	14
1.1.4 Gli Elementi di CALMS nel Contesto della QA	15
1.1.5 I Tre Modi di DevOps	16
1.1.6 Benefici, Rischi e Insidie di DevOps	17
1.2 Implementare DevOps in un'Organizzazione	18
1.2.1 Ruoli in un Team DevOps	18
1.2.2 Site Reliability Engineering (SRE)	19
1.2.3 Pattern e Anti-pattern dei Team DevOps	20
2 Quality Assurance (QA) e Testing in DevOps - 360 minuti	22
2.1 Contributo al Value Stream a Supporto del Quality Engineering (QE)	23
2.1.1 QA e Testing in un Contesto DevOps	23
2.1.2 Confrontare gli Obiettivi del Test nei Diversi Modelli di SDLC	24
2.1.3 Continuous Testing	25
2.1.4 Pull Request	26
2.2 Implementazione del Loop DevOps	27
2.2.1 QA e Testing nella Continuous Discovery	27
2.2.2 QA e Testing nella CI	28
2.2.3 QA e Testing nella CD	29
2.2.4 QA e Testing nel Continuous Deployment	29
2.2.5 QA e Testing nel Release on Demand	30
2.2.6 Implementazione della Pipeline di CI/CD	31

Quality in DevOps (CT-QDO) Syllabus

3	Attività Automatizzate e Manuali in DevOps - 510 minuti	34
3.1	Il Supporto dell'Automazione alla Quality Assurance (QA) in DevOps	36
3.1.1	Single Source of Truth (SSOT) per il Testware	36
3.1.2	Tracciabilità tra la Base di Test e il Testware	37
3.1.3	Reporting della Qualità per una Pipeline di CI/CD	37
3.1.4	Automazione della Gestione dei Dati di Test	39
3.1.5	Analisi Statistica dei Risultati dei Test	40
3.1.6	Gestione degli Ambienti di Test Standardizzata, Controllata e Automatizzata	40
3.2	Testing Automatizzato nei Team DevOps	41
3.2.1	Regression Testing nella Pipeline di CI/CD	41
3.2.2	Punti Chiave del Testing delle API	42
3.2.3	Confrontare la Test Automation nei Diversi Modelli di SDLC	43
3.3	Testing Manuale nei Team DevOps	44
3.3.1	Esempi di Testing Manuale	45
3.3.2	Testing Esplorativo nella Pipeline di CI/CD	45
3.3.3	Crowd Testing	46
3.3.4	Eventi di Quality Hunting	47
4	Strumenti e Pratiche in DevOps - 200 minuti	48
4.1	Strumenti a Supporto di DevOps all'Interno dell'Organizzazione	49
4.1.1	Capability degli Strumenti	49
4.1.2	Strumenti a Supporto della Quality Assurance (QA) e del Testing	51
4.2	Tecnologie e Pratiche a Supporto della Qualità in DevOps	52
4.2.1	Attività non di Testing all'Interno di una Pipeline di CI/CD	52
4.2.2	Principali Strategie di Rilascio	53
4.2.3	Infrastructure as code (IaC)	54
4.2.4	Feature Toggle	54
4.2.5	Strategie di Branch	55
4.2.6	Chaos Engineering	56
4.2.7	Telemetria e Observability	56
4.2.8	Software Bill of Materials (SBOM)	57
4.2.9	Containerizzazione	58
5	Termini specifici di Quality in DevOps	59
6	Trademark	62
7	Appendice A – Obiettivi di Apprendimento/Livello Cognitivo di Conoscenza	63
8	Appendice B – Matrice di Tracciabilità dei Business Outcome rispetto agli Obiettivi di Apprendimento	65
9	Appendice C – Note di Rilascio	70
10	Riferimenti	71
11	Ulteriori Letture	75
12	Appendice D – Indice Analitico	77

Ringraziamenti

Questo documento è stato rilasciato ufficialmente dall'Assemblea Generale di ISTQB® il 17 aprile 2026.

È stato prodotto da un team dell'International Software Testing Qualifications Board: Vipul Kocher (presidente dello Specialist Technologies and Approaches Working Group), Tamás Horváth (vicepresidente dello Specialist Technologies and Approaches Working Group), Szilárd Széll (product owner), Kari Kakkonen, Rik Marselis, Katja Obring, Tal Pe'er, Daniel Poľan, Yaron Tsubery, Tomas Tumasonis, Aneta Derková

Le seguenti persone, in ordine alfabetico, sono gli autori di questo syllabus: Kari Kakkonen, Rik Marselis, Katja Obring, Szilárd Széll, Yaron Tsubery.

Le seguenti persone, in ordine alfabetico, hanno dato un contributo importante a questo syllabus: Mantas Anilius, Aneta Derkova, Matthias Hamburg, Tamas Horvath, Vipul Kocher, Gary Mogyorodi, Tal Pe'er, Daniel Poľan, Patrick Quilter, Jean-Francois Riverin, Tomas Tumasonis e Galit Zucker.

Il team ringrazia le seguenti organizzazioni per il contributo dato al contenuto di questo syllabus: ASTQB, AT*SQA, DevOps United.

Il team ringrazia Gary Mogyorodi per l'editing tecnico, il team di review e i Member Board per i loro suggerimenti e contributi.

Le seguenti persone hanno partecipato alla review, ai commenti e alla votazione di questo syllabus: Mariam Abdellatif, May Abu-Sbeit, Gergely Agnecz, Amer Alatrash, Laura Albert, Chris van Bael, Jürgen Beniermann, Earl Burba, Tamás Csákó, Wim Decoutere, Walter Eder, Raymond Gillespie, Ole Christian Hansen, Josephine Irungu, Mattijs Kemmink, John Kurowski, Jędrzej Kwapiński, Anders Larsen, Roberta Lippelli, Sebastian Malyska, Vincenzo Marrazzo, Gary Mogyorodi, Frank Neiryndck, Ingvar Nordström, Giorgio Pisani, Andrew Pollner, Nishan Portoyan, Meile Posthuma, Patrick Quilter, Miroslav Renda, Piet de Roo, Adam Roman, Jan Sabak, Adam Scierski, Márton Siska, Péter Sótér, Benjamin Timmermans, Giancarlo Tomasig, Stephanie Ulrich, Linda Vreeswijk, Dominik Weber, Geoffrey Wemans, Claude Zhang

Allineamento specifico

TMMi® e ISTQB® hanno stretto un'alleanza per promuovere insieme la professione del Software Testing. Durante lo sviluppo di questo documento il comitato tecnico TMMi® e il gruppo di lavoro del syllabus ISTQB® Certified Tester Quality in DevOps hanno collaborato. TMMi® è un modello di miglioramento del testing. Fornisce un approccio di miglioramento predefinito con priorità basate sulla struttura del modello TMMi®. Nel documento «TMMi® in the DevOps world» (E. v. Veenendaal, 2025) l'attenzione è posta sull'interpretazione degli obiettivi di miglioramento del TMMi® per chi opera in un contesto DevOps. Tale documento non fornisce una descrizione dettagliata delle buone pratiche di ingegneria DevOps. Il syllabus ISTQB® Certified Tester Quality in DevOps è invece un documento basato sui contenuti. Il suo scopo è fornire una descrizione dettagliata delle buone pratiche di ingegneria DevOps, per esempio delle attività di test. I due documenti sono quindi fortemente complementari. TMMi® individua quali processi necessitano di miglioramento, mentre il syllabus ISTQB® Certified Tester Quality in DevOps delinea le best practice ingegneristiche per l'implementazione.

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

0 Introduzione

0.1 Scopo di questo Syllabus

Questo syllabus costituisce la base della International Software Testing Qualification per il syllabus Certified Tester Quality in DevOps (CT-QDO). ISTQB® mette a disposizione questo syllabus come segue:

1. Ai Member Board, per tradurlo nella propria lingua locale e per accreditare i training provider. I Member Board possono adattare il syllabus alle specifiche esigenze linguistiche e modificare i riferimenti per adattarli alle pubblicazioni locali.
2. Agli organismi di certificazione, per derivare le domande d'esame nella lingua locale, adattate agli obiettivi di apprendimento di questo syllabus.
3. Ai training provider, per produrre il materiale didattico e definire metodi di insegnamento appropriati.
4. Ai candidati alla certificazione, per prepararsi all'esame di certificazione (sia nell'ambito di un corso di formazione sia autonomamente).
5. Alla comunità internazionale dell'ingegneria del software e dei sistemi, per far progredire la professione del testing del software e dei sistemi, e come base per libri e articoli.

0.2 Il Certified Tester Quality in DevOps

La certificazione ISTQB® Certified Tester Quality in DevOps (CT-QDO) supporta le persone coinvolte nello sviluppo software basato su DevOps nell'applicare adeguate attività di qualità e di test per raggiungere il giusto livello di qualità delle proprie soluzioni.

La certificazione CT-QDO è pensata per i professionisti impegnati nella qualità e nel testing all'interno di un ambiente DevOps. L'idea è offrire agli specialisti DevOps l'opportunità di acquisire una certificazione riconosciuta a livello internazionale in questo ambito e di affermare i vantaggi specifici degli specialisti certificati della qualità in DevOps attraverso competenze specifiche nella metodologia di qualità e di testing in DevOps.

DevOps è diventato la modalità prevalente dello sviluppo software moderno in molti domini. Estende i concetti di autonomia, consegne frequenti e attenzione al feedback anticipato del cliente propri dello sviluppo software Agile, per raggiungere la giusta qualità del software. Quality Assurance (QA) e testing hanno un forte rilievo nella cultura DevOps. In un contesto DevOps il quality engineering (QE), che comprende QA e testing, si svolge durante tutte le fasi del ciclo di vita dello sviluppo software (SDLC). DevOps copre concetti come continuous delivery (CD) e release on demand, che i tester del software dovrebbero comprendere per sviluppare pratiche efficaci di qualità e di test.

0.3 Percorso di Carriera per i Tester

Lo schema ISTQB® supporta i professionisti del testing in tutte le fasi della loro carriera, offrendo sia ampiezza sia profondità di conoscenza.

Quality in DevOps (CT-QDO) Syllabus

Il Certified Tester Quality in DevOps si fonda sulla qualificazione di ISTQB® Certified Tester Foundation Level (*ISTQB® CTFL*, v4.0). I syllabus ISTQB® relativi ad Agile e quelli relativi alla test automation sono utili per la comprensione di questo syllabus. Mentre il syllabus ISTQB® Certified Tester Foundation Level fornisce le conoscenze e le competenze di base nel testing del software, i syllabus ISTQB® relativi ad Agile ampliano il syllabus Certified Tester Foundation Level e spiegano come si svolge il testing in un team Agile, e i syllabus ISTQB® relativi alla test automation spiegano come la test automation viene costruita e utilizzata per il testing.

Poiché questo syllabus si concentra sulla cultura DevOps, amplia il portafoglio di prodotti ISTQB® verso DevOps, direzione in cui si sta muovendo lo sviluppo software, con benefici quali un time to market più rapido, clienti più soddisfatti e una qualità migliore.

Chi consegue la certificazione ISTQB® CT-QDO potrebbe essere interessato anche alle altre certificazioni ISTQB® Specialist, in particolare a ISTQB® Certified Tester Foundation Level Agile Tester (*ISTQB® CT-AT*, v1.1), ISTQB® Certified Tester Agile Technical Tester (*ISTQB® CT-ATT*, v1.1) e ISTQB® Certified Tester Agile Test Leadership at Scale (*ISTQB® CT-ATLaS*, v2.0), oltre al syllabus ISTQB® Certified Tester Test Automation Strategy (*ISTQB® CT-TAS*, v1.0) e anche a ISTQB® Certified Tester Test Automation Engineer (*ISTQB® CTAL-TAE*, v2.0).

0.4 Business Outcome

Questa sezione elenca i business outcome attesi da un candidato che ha conseguito la certificazione CT-QDO.

Una persona certificata CT-QDO è in grado di:

Codice	Descrizione
QDO-BO1	Spiegare Come la Quality Assurance è Supportata dai Concetti DevOps e Vi Contribuisce
QDO-BO2	Comprendere i Concetti dell'Implementazione di DevOps in un'Organizzazione
QDO-BO3	Contribuire a Tutte le Fasi del Value Stream a Supporto del Quality Engineering
QDO-BO4	Contribuire all'Implementazione del Loop DevOps
QDO-BO5	Descrivere Come l'Automazione Supporta la Quality Assurance in DevOps
QDO-BO6	Implementare il Testing Automatizzato nei Team DevOps
QDO-BO7	Implementare il Testing Manuale nei Team DevOps
QDO-BO8	Selezionare gli Strumenti a Supporto di DevOps all'Interno dell'Organizzazione
QDO-BO9	Comprendere Tecnologie e Pratiche a Supporto della Qualità in DevOps

0.5 Obiettivi di Apprendimento, Obiettivi Pratici e Livello Cognitivo di Conoscenza

Gli obiettivi di apprendimento e gli obiettivi pratici supportano i business outcome e sono utilizzati per costruire gli esami Certified Tester Quality in DevOps (CT-QDO).

In generale tutti i contenuti di questo syllabus sono esaminabili, a eccezione dell'Introduzione, degli Obiettivi Pratici e delle Appendici. Le domande d'esame verificheranno la conoscenza delle parole chiave a livello K2 (vedi sotto) o degli obiettivi di apprendimento al rispettivo livello di conoscenza. Gli specifici obiettivi di apprendimento e i relativi livelli di conoscenza sono riportati all'inizio di ogni capitolo e sono classificati come segue:

v1.0 Versione di Rilascio

Quality in DevOps (CT-QDO) Syllabus

- K1: Ricordare
- K2: Comprendere
- K3: Applicare

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

Per tutti i termini elencati come parole chiave subito sotto i titoli dei capitoli si deve conoscere (K2) il nome corretto e la definizione riportati nel Glossario ISTQB® (ISTQB, 2025), anche se non esplicitamente menzionati nell'obiettivo di apprendimento.

Gli specifici Obiettivi Pratici sono riportati all'inizio di ogni capitolo. Il livello di un HO è classificato come segue:

- H0: può includere una dimostrazione dal vivo di un esercizio o un video registrato. Poiché non viene svolto dal discente, non è propriamente un esercizio.
- H1: esercizio guidato. I discenti seguono una sequenza di passi svolti dal docente.
- H2: esercizio con suggerimenti. Il discente riceve un esercizio con suggerimenti per risolverlo nel tempo assegnato.
- H3: esercizi non guidati, senza suggerimenti.

0.6 L'Esame per la Certificazione Certified Tester Quality in DevOps

L'esame per il certificato CT-QDO si baserà su questo syllabus. Le risposte alle domande d'esame possono richiedere l'uso di materiale tratto da più di una sezione di questo syllabus. Tutte le sezioni del syllabus sono esaminabili, a eccezione dell'Introduzione, degli Obiettivi Pratici e delle Appendici. Standard e libri sono inclusi come riferimenti, ma il loro contenuto non è esaminabile oltre a quanto riassunto nel syllabus stesso a partire da tali standard e libri.

Per ulteriori dettagli si rimanda al documento Exam Structures and Rules (*ISTQB® Exam Structures and Rules*, v1.2).

I criteri di ingresso per sostenere l'esame ISTQB® Certified Tester Quality in DevOps sono che i candidati possiedano

- un certificato ISTQB® Certified Tester Foundation Level
- un interesse per il testing del software e per DevOps

Ai candidati si raccomanda inoltre vivamente di:

- avere almeno un'esperienza minima nello sviluppo software o nel testing del software, per esempio sei mesi di esperienza come tester di sistema o di accettazione utente, oppure come sviluppatore software;
- 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 i training provider e i titolari di materiale formativo il cui materiale di corso segue questo syllabus. I training provider dovrebbero ottenere le linee guida per l'accreditamento dal Member Board o dall'organismo che effettua l'accreditamento. Un corso accreditato è riconosciuto come conforme a questo syllabus e può includere un esame ISTQB® come parte del corso.

Le linee guida per l'accreditamento di questo syllabus seguono le Accreditation Guidelines generali (ISTQB® *Generic Accreditation Guidelines*, v2.4) pubblicate dal Processes Management and Compliance Working Group di ISTQB®.

0.8 Gestione degli Standard

Organizzazioni internazionali di standardizzazione come IEEE e ISO hanno emesso 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 framework (come nei riferimenti a ISO 25010 relativi alle caratteristiche di qualità) oppure fornire una fonte di informazioni aggiuntive se il lettore lo desidera.

Si noti che i syllabus ISTQB® utilizzano i documenti di standard come riferimento. I documenti di standard non sono destinati all'esame. Per maggiori informazioni sugli standard si rimanda al Capitolo *Capitolo 10*.

0.9 Livello di Dettaglio

Il livello di dettaglio di questo syllabus consente corsi ed esami coerenti a livello internazionale. Per raggiungere questo obiettivo il syllabus si compone di:

- obiettivi didattici generali che descrivono l'intento del syllabus Certified Tester Quality in DevOps;
- un elenco di termini che i discenti devono essere in grado di richiamare;
- obiettivi di apprendimento per ciascuna area di conoscenza, che descrivono il risultato cognitivo di apprendimento da conseguire;
- una descrizione dei concetti chiave, con riferimenti a fonti quali letteratura riconosciuta o standard.

Il contenuto del syllabus non è una descrizione dell'intera area di conoscenza del testing del software: riflette il livello di dettaglio da trattare nei corsi di formazione CT-QDO.

Il syllabus utilizza la terminologia (il nome e il significato) dei termini usati nella QA e nel testing secondo il Glossario ISTQB®.

0.10 Com'è Organizzato questo Syllabus

Vi sono quattro capitoli con contenuti esaminabili. Il titolo di primo livello di ciascun capitolo indica il tempo previsto per il capitolo; i tempi non sono indicati al di sotto del livello di capitolo. Per i corsi di formazione accreditati il syllabus richiede un minimo di 20,2 ore di docenza distribuite su almeno tre giorni. Il tempo minimo di formazione distribuito sui quattro capitoli è il seguente:

- Capitolo 1: 140 minuti, Fondamenti di DevOps

Quality in DevOps (CT-QDO) Syllabus

- Spiegare Come la Quality Assurance è Supportata dai Concetti DevOps e Vi Contribuisce
 - Comprendere i Concetti dell'Implementazione di DevOps in un'Organizzazione
- Capitolo 2: 360 minuti, Quality Assurance e Testing in DevOps
 - Contribuire a Tutte le Fasi del Value Stream a Supporto del Quality Engineering
 - Contribuire all'Implementazione del Loop DevOps
- Capitolo 3: 510 minuti, Attività Automatizzate e Manuali in DevOps
 - Descrivere Come l'Automazione Supporta la Quality Assurance in DevOps
 - Implementare il Testing Automatizzato nei Team DevOps
 - Implementare il Testing Manuale nei Team DevOps
- Capitolo 4: 200 minuti, Strumenti e Pratiche in DevOps
 - Selezionare gli Strumenti a Supporto di DevOps all'Interno dell'Organizzazione
 - Comprendere Tecnologie e Pratiche a Supporto della Qualità in DevOps

1 Fondamenti di DevOps - 140 minuti

Parole chiave

Quality Assurance, testing

Parole chiave specifiche di Quality in DevOps

CALMS, change fail percentage, change lead time, team DevOps cross-funzionale, frequenza di deploy, DevOps, tempo di recupero da un deploy fallito, ciclo di feedback, flusso, Service Level Agreement, Service Level Indicator, Service Level Objective, Site Reliability Engineering, value stream, wall of confusion

Obiettivi di Apprendimento del Capitolo 1:**1.1 Spiegare Come la Quality Assurance è Supportata dai Concetti DevOps e Vi Contribuisce**

- QDO-1.1.1 (K1) Richiamare i concetti chiave di DevOps
- QDO-1.1.2 (K1) Richiamare i concetti relativi al wall of confusion in DevOps
- QDO-1.1.3 (K2) Spiegare le metriche DORA di delivery performance e operational performance
- QDO-1.1.4 (K2) Differenziare gli elementi di CALMS in termini di Quality Assurance
- QDO-1.1.5 (K2) Spiegare come la Quality Assurance supporta i tre modi di DevOps
- QDO-1.1.6 (K2) Spiegare i benefici, i rischi e le insidie di DevOps

1.2 Comprendere i Concetti dell'Implementazione di DevOps in un'Organizzazione

- QDO-1.2.1 (K2) Distinguere i ruoli del team DevOps
- QDO-1.2.2 (K2) Spiegare il concetto di Site Reliability Engineering (SRE) in relazione a DevOps
- QDO-1.2.3 (K2) Distinguere i diversi pattern e anti-pattern dei team DevOps

Obiettivi Pratici:**1.2 Implementare DevOps in un'Organizzazione**

- QDO-1.2 (H0) Riassumere Come i Principi DevOps Sono Visibili in un Caso di Studio

1.1 Quality Assurance (QA) in DevOps

DevOps è la combinazione di filosofie culturali, pratiche e strumenti che accrescono la capacità di un'organizzazione di rilasciare applicazioni e servizi ad alta velocità, facendo così evolvere e migliorare i prodotti a un ritmo più rapido rispetto alle organizzazioni che utilizzano modelli di sviluppo sequenziali e processi di gestione dell'infrastruttura (Amazon, 2024).

DevOps si è imposto all'attenzione nel 2008, quando Patrick Debois (Debois, 2011) invitò persone con interessi affini a una conferenza per discutere di come la consegna del software, e dell'infrastruttura, potesse utilizzare lo sviluppo software Agile. Il termine DevOps è emerso dopo che la Velocity Conference del 2009 introdusse i concetti di consegna rapida, frequente e di successo (Debois, 2011).

Questo capitolo tratta i concetti DevOps e il modo in cui il quality engineering (QE), che comprende Quality Assurance (QA) e testing, si rapporta a essi. La QA comprende tutte le attività volte a fornire fiducia nel fatto che i requisiti di qualità saranno soddisfatti, cioè lungo l'intero ciclo di vita dello sviluppo software (SDLC). Il QE comprende tutte le attività relative alla qualità che contribuiscono a creare software rispondente alle esigenze del cliente. DevOps considera lo sviluppo software come un value stream che guida il software dal concetto al lancio sul mercato (*ISTQB® CT-ATLaS, v2.0*). Questo capitolo spiega inoltre i team DevOps, intendendo con ciò qualsiasi team che applichi pratiche DevOps.

1.1.1 Concetti Chiave di DevOps

DevOps sta per «Development and Operations». Si riferisce a persone che lavorano insieme per costruire, rilasciare ed esercire il software. In DevOps il processo di sviluppo software dovrebbe considerare, oltre alla scrittura del codice, anche altri aspetti dello sviluppo quali l'analisi, la continuous integration (CI), il testing funzionale, il testing non-funzionale — in particolare la sicurezza — il deployment, la delivery, il release, la gestione dell'infrastruttura, la manutenzione e il monitoraggio in produzione. Il successo dello sviluppo software dovrebbe essere misurato in base ai risultati della consegna e all'utilizzo di software della giusta qualità.

DevOps è una cultura che promuove collaborazione, comunicazione e miglioramento continuo, più che un framework, uno standard, un ruolo, una funzione, un team separato, uno strumento o un obiettivo (DASA, 2024). DevOps sostiene con forza l'automazione e il monitoraggio in tutte le fasi della consegna del software. Ciò si concretizza spesso sotto forma di pipeline di continuous integration/continuous delivery (CI/CD). DevOps promuove autonomia, leadership, collaborazione e innovazione, guidando al contempo il successo del business attraverso cicli di consegna più brevi, una maggiore frequenza di deploy e release affidabili e di alta qualità che soddisfano le esigenze del cliente.

Le pratiche DevOps si basano sui seguenti principi (DASA, 2019):

1. Azione centrata sul cliente: avere cicli di feedback brevi con clienti reali.
2. Creare avendo in mente il risultato finale: concentrarsi esplicitamente sulla costruzione di prodotti funzionanti consegnati ai clienti.
3. Responsabilità end-to-end: creare team DevOps che progettano, costruiscono, testano ed esercano il software.
4. Team autonomi e cross-funzionali: creare team DevOps con competenze diversificate, testing incluso, in modo che siano autonomi.

Quality in DevOps (CT-QDO) Syllabus

5. Miglioramento continuo: abbracciare una cultura del miglioramento quotidiano del prodotto, del processo e delle competenze delle persone, per adattarsi in base al feedback della sperimentazione.
6. Automatizzare tutto il possibile: puntare a una pipeline di CI/CD automatizzata che rilasci con tempi di ciclo brevi.

DevOps è spesso rappresentato come un loop a forma di infinito, con passi sia manuali sia automatizzati. Il tipico loop DevOps comprende diversi passi che possono essere raggruppati in continuous discovery, CI, CD, continuous deployment e release on demand. È descritto nella *Sezione 2.2* ed esistono molteplici varianti del loop DevOps.

1.1.2 Abbattere il Wall of Confusion

Il wall of confusion è una metafora della separazione tra i team di sviluppo e quelli operativi nei modelli di sviluppo sequenziali. Tradizionalmente QA e testing si trovano dal lato del muro occupato dal team di sviluppo.

Mentre il team di sviluppo punta a modificare continuamente il software per rilasciare nuove funzionalità, le operations preferirebbero la stabilità, minimizzando così il numero di cambiamenti in produzione. Ne derivano silos con obiettivi in conflitto, mancanza di comunicazione, processi manuali e ambienti non consistenti. In breve, vi sono mentalità (cioè motivazioni e obiettivi), processi e strumenti differenti. Le differenze sono tanto più marcate quanto più i team sono isolati in silos. In DevOps il wall of confusion viene abbattuto integrando i team affinché lavorino insieme, migliorando la comunicazione e aumentando la collaborazione. Serve leadership per migliorare o eliminare processi, attività e pratiche inefficienti (per esempio prevenire una cultura della colpa mettendo l'accento sulla responsabilità condivisa) (DASA, 2024).

QA e test dovrebbero essere svolti lungo tutto il processo, sia sul versante Dev sia su quello Ops, rendendo il testing un fattore chiave per l'abbattimento del muro.

L'adozione di DevOps può abbattere i muri tra i vari team isolati in silos (per esempio sicurezza, analisi di business o usabilità).

1.1.3 Le Metriche di Performance DORA

Il programma DevOps Research & Assessment (DORA) pubblica annualmente lo State of DevOps Report. La ricerca (DORA, 2024) mostra quattro metriche di delivery performance che offrono un modo efficace di misurare il processo di consegna del software.

Due metriche riguardano il throughput (cioè la velocità con cui si effettuano i cambiamenti):

- Change lead time: la durata che intercorre dal commit del codice fino al deployment in produzione andato a buon fine. Riflette l'efficienza del processo di consegna e tende a essere breve nei team ad alte prestazioni.
- Frequenza di deploy: la frequenza dei cambiamenti software rilasciati in produzione. Riflette quanto il processo di consegna sia agile e reattivo e tende a essere elevata nei team ad alte prestazioni.

Due metriche riguardano la stabilità (cioè la qualità dei cambiamenti consegnati e la capacità del team DevOps di correggere i failure):

v1.0 Versione di Rilascio

Quality in DevOps (CT-QDO) Syllabus

- Change fail percentage: la percentuale di deployment che causano failure in produzione, richiedendo hotfix o rollback. Indica quanto sia affidabile il processo di consegna e tende a essere ridotta nei team ad alte prestazioni.
- Tempo di recupero da un deploy fallito: il tempo necessario per ripristinare la situazione dopo un deployment fallito. Indica quanto l'organizzazione sia resiliente e reattiva e tende a essere breve nei team ad alte prestazioni.

I team DevOps ad alte prestazioni tendono a ottenere buoni risultati su tutte e quattro le metriche. Per farlo hanno bisogno di buone pratiche di QE, tra cui test automation, automazione CI/CD, collaborazione, approcci test-first, un approccio whole-team (*ISTQB® CTFL*, v4.0), strategie di deployment ben scelte e monitoraggio continuo. Più il value stream è veloce, meglio è.

DORA ha aggiunto l'affidabilità come indicatore di operational performance. Secondo DORA l'affidabilità si compone di metriche o caratteristiche di qualità quali disponibilità, latenza, efficienza delle prestazioni e scalabilità. I team DevOps ottengono migliori risultati di delivery performance (per esempio un minor burnout del personale) concentrandosi sull'affidabilità (DORA, 2022). Le quattro metriche di delivery performance riguardano il processo, mentre l'affidabilità riguarda il software in uso.

QA e testing contribuiscono direttamente a una migliore affidabilità, in particolare attraverso approcci test-first, testing non-funzionale, chaos engineering (vedi *Sezione 4.2.6*) e quality hunting (vedi *Sezione 3.3.4*).

1.1.4 Gli Elementi di CALMS nel Contesto della QA

CALMS sta per culture, automation, lean, measurement e sharing (Kim; Humble, et al., 2016). È un framework che valuta la capacità dell'organizzazione di adottare processi DevOps e di misurare il successo durante una trasformazione DevOps.

Culture riflette il fatto che cambiare solo processi e tecnologia non è sufficiente per produrre un cambiamento duraturo. È essenziale una cultura collaborativa e orientata alla soluzione dei problemi. La cultura DevOps è un'estensione dello sviluppo software Agile, in cui i team Agile hanno l'autonomia di consegnare valore ai clienti pur ricercando la necessaria collaborazione con gli altri team. Questa autonomia comprende il testing e un approccio whole-team alla titolarità della qualità, cosicché il testing non è esterno al team DevOps (Crispin et al., 2008).

Automation si riferisce all'automatizzare quanto più possibile nella pipeline di CI/CD. Ciò comprende build, deployment, provisioning, automazione dei processi, analisi statica e test automation, quindi «everything as code». L'automazione nelle pipeline di CI/CD consente al team DevOps di far passare continuamente piccoli cambiamenti mantenendo una buona qualità e accelerando il ciclo di feedback.

Lean (Humble; Molesky, et al., 2020) significa snellire i processi ed eliminare gli sprechi (per esempio i passi non necessari del processo) affinché il processo sia il più efficiente possibile. Il miglioramento continuo fa parte del lean e si applica anche alle attività di test, che non dovrebbero diventare colli di bottiglia per il flusso del lavoro. Occorre quindi ripensare quando e come si svolge il testing nel processo.

Measurement fornisce dati per le decisioni su come il software viene utilizzato e su come funziona l'SDLC. La misurazione comporta l'uso di metriche azionabili su qualità ed efficienza per migliorare i prodotti e per migliorare con continuità l'SDLC.

Sharing riguarda la cultura aperta della discussione delle sfide e dei successi dello sviluppo software nei team DevOps. Riguarda anche la condivisione di buone pratiche e idee ai fini dell'apprendimento e del

miglioramento continuo. QA e testing svolgono un ruolo essenziale nella creazione di informazioni da condividere (per esempio il feedback del testing automatizzato relativo a funzionalità e testabilità).

Ogni elemento di CALMS rafforza il ruolo di QA e testing nel garantire che la qualità sia incorporata in ogni fase dell'SDLC, consentendo una consegna più rapida di software di alta qualità e riducendo al contempo i rischi.

Esistono altre versioni di CALMS (per esempio CALMR in SAFe (*Scaled Agile Framework*, 2023a), dove R sta per recovery).

1.1.5 I Tre Modi di DevOps

Per capire come la QA si inserisca nelle pratiche di sviluppo software è essenziale esaminare i principi DevOps. DevOps rappresenta un movimento culturale e tecnico che mira a unificare sviluppo e operations, ma i suoi principi fondamentali si estendono ben oltre questi due soli ambiti. Tali principi, noti come i tre modi di DevOps, forniscono un framework che aiuta le organizzazioni a trasformare il proprio processo di consegna del software mantenendo elevati standard di qualità. Comprendere questi principi spiega come le pratiche di QA evolvono e si integrano in un contesto DevOps.

In «The Phoenix Project» (Kim; Behr, et al., 2018) i tre modi di DevOps sono i seguenti:

1. Il primo modo: flusso (flow)
Questo principio si concentra sul flusso regolare del lavoro dallo sviluppo alle operations, assicurando che il lavoro si muova rapidamente ed efficientemente attraverso il processo. Il flusso mette in evidenza l'importanza di minimizzare i momenti di passaggio di consegne, di ridurre le dimensioni dei lotti e di rimuovere i colli di bottiglia per creare un processo snello e prevedibile.
2. Il secondo modo: ciclo di feedback
Questo principio pone l'accento sulla creazione di cicli di feedback rapidi e continui. Ciò consente di individuare e risolvere rapidamente i problemi, evitando che si aggravino. Incoraggia inoltre una cultura in cui tutti possono imparare dagli errori e migliorare con continuità il prodotto e il processo di consegna.
3. Il terzo modo: continuous learning and experimentation
Questo principio evidenzia l'importanza di alimentare una cultura di miglioramento continuo e di sperimentazione. Incoraggia l'innovazione, l'assunzione di rischi calcolati e l'apprendimento sia dai successi sia dai failure. Questo principio sostiene l'idea che l'apprendimento continuo sia essenziale per adattarsi ai cambiamenti e migliorare il sistema nel suo complesso.

La QA è parte integrante dei tre modi di DevOps:

Flow:

La QA consente una progressione regolare del codice attraverso approcci test-first.

Feedback:

La QA crea cicli di feedback rapidi tramite test automation continua, ambienti di pre-produzione standardizzati e monitoraggio per raccogliere dati d'uso reali a fini di miglioramento.

Continuous Learning:

La QA favorisce l'innovazione attraverso testing mirato delle nuove funzionalità, eventi di qualità organizzati come il quality hunting e regression testing esteso.

1.1.6 Benefici, Rischi e Insidie di DevOps

DevOps porta opportunità, ma le organizzazioni dovrebbero prestare attenzione ai rischi e alle insidie quando lo implementano.

Benefici

DevOps può restituire benefici significativi alle organizzazioni (DORA, 2024). Tra questi:

- mettere i team DevOps nelle condizioni di valutare la qualità in ogni fase di sviluppo, deployment e produzione;
- raccogliere feedback precocemente e frequentemente per migliorare sia il software sia l'SDLC;
- consegnare puntualmente con costi inferiori grazie a software di qualità più elevata e a processi più efficienti;
- rispondere più rapidamente alle esigenze che cambiano;
- ripristinare più rapidamente la situazione dopo i failure in produzione;
- migliorare il time to market impiegando processi e comunicazione efficienti ed efficaci;
- rendere più rapida la scalabilità del software al crescere del numero di utenti grazie a infrastructure as code (IaC) (vedi *Sezione 4.2.3*);
- migliorare la collaborazione nei team DevOps lungo tutto l'SDLC;
- ridurre lo stress dei dipendenti consentendo release più frequenti di piccoli incrementi di funzionalità;
- ridurre i problemi con fornitori e terze parti aumentando collaborazione e trasparenza in tutte le attività del team DevOps;
- migliorare il ritorno sull'investimento della test automation (Humble; Farley, 2010).

Rischi e Insidie

Se i benefici di DevOps sono piuttosto evidenti, molte cose possono andare storte. I team DevOps dovrebbero prevedere questi rischi affinché non diventino insidie tali da far fallire la trasformazione DevOps. Rischi e insidie tipici comprendono:

- mancanza di supporto da parte del management;
- eliminare o sostituire le Ops invece di trasformarle;
- sostituire lo sviluppo software Agile invece di estenderlo;
- credere che esista un solo modo di fare DevOps;
- usare DevOps solo come titolo professionale;
- creare un team DevOps ignorando però la necessità di cambiamento negli altri team e nell'organizzazione;
- ridurre il numero di persone aumentando l'automazione invece di rendere le persone più efficienti;

Quality in DevOps (CT-QDO) Syllabus

- introdurre solo gli strumenti della pipeline di CI/CD dimenticando il cambiamento culturale DevOps;
- implementare una pipeline di CI/CD con poca o nessuna test automation;
- usare solo testing automatizzato là dove il testing manuale sarebbe vantaggioso.

1.2 Implementare DevOps in un'Organizzazione

DevOps può essere implementato in vari modi all'interno di un'organizzazione. Indipendentemente da come viene implementato, vi è sempre bisogno di QE, testing incluso. I membri del team devono comprendere questi concetti per lavorarci efficacemente.

Questa sezione tratta i ruoli tipici dei team DevOps, comprese le varianti del ruolo di tester, il Site Reliability Engineering (SRE) in DevOps e le topologie, i pattern e gli anti-pattern di team che influiscono sul successo di DevOps. Si considera inoltre l'esperienza dei membri del team in relazione all'ambiente di sviluppo, indicata come developer experience (DevEx).

Obiettivo Pratico QDO-1.2 (H0) Riassumere Come i Principi DevOps Sono Visibili in un Caso di Studio

Per questo obiettivo pratico l'esercizio di livello H0 supporta i discenti nella preparazione agli obiettivi pratici successivi fornendo il contesto di un caso di studio. L'organizzazione immaginaria descritta nel caso di studio va considerata solo come un'organizzazione parzialmente DevOps, in modo che gli esercizi basati sugli obiettivi pratici successivi possano implementare miglioramenti ai modi di lavorare all'interno di questa organizzazione.

1.2.1 Ruoli in un Team DevOps

Un ruolo è un insieme di responsabilità e di attività, e non necessariamente un titolo professionale. I ruoli principali in un team DevOps cross-funzionale sono business analyst, sviluppatore, tester e operations engineer, spesso supportati da un product owner e da uno scrum master. Tutti i membri del team sono responsabili insieme del QE.

I membri di un team DevOps cross-funzionale possono scegliere qualsiasi attività coerente con le proprie conoscenze e competenze. Ciò significa che una persona può ricoprire ruoli diversi nel tempo e può anche svolgere più ruoli in parallelo. I membri del team possono collaborare in coppia su un'attività. Questo approccio ottimizza il flusso di lavoro nel team DevOps.

Un team DevOps cross-funzionale possiede tutte le conoscenze e le competenze necessarie per svolgere qualsiasi attività comune per il team DevOps. Nessuna parte delle conoscenze e delle competenze dovrebbe risiedere in un solo membro del team. In questo modo un team DevOps cross-funzionale può continuare a funzionare quando uno dei suoi membri è temporaneamente indisponibile.

Tuttavia alcune attività specialistiche possono eccedere le competenze disponibili nei team DevOps (per esempio la configurazione e la manutenzione della piattaforma). Per questa ragione le organizzazioni possono scegliere di dotarsi di specifici team di piattaforma che forniscono strumenti self-service e un'infrastruttura affidabile che astrae le attività operative complesse, consentendo ai team DevOps una

consegna del software più rapida e consistente. In questo modo il team di piattaforma riduce l'onere operativo sui team DevOps e consente deployment più rapidi, sicuri e affidabili.

La qualità dei prodotti di lavoro del team DevOps è mantenuta attraverso collaborazione, apprendimento continuo e miglioramento. Ogni prodotto di lavoro è gestito da almeno due membri del team. Il processo di review lo garantisce (per esempio con le pull request (PR)) (vedi *Sezione 2.1.4*). Se il prodotto di lavoro non viene accettato, il reviewer fornisce feedback affinché l'autore possa migliorarlo, autonomamente o in collaborazione con altri membri del team. Qualsiasi membro del team può essere autore, contributore o reviewer dei prodotti di lavoro.

Un team DevOps cross-funzionale composto da un gruppo eterogeneo di persone competenti e motivate, che collaborano e si assumono insieme la responsabilità, è un modo efficace per creare e mantenere il software. Questo è il fondamento della cultura DevOps.

1.2.2 Site Reliability Engineering (SRE)

L'SRE è un insieme di pratiche che applicano aspetti dell'ingegneria del software alle operations e all'infrastruttura IT. L'SRE ha origine nell'organizzazione Google (Beyer et al., 2016). L'SRE si concentra sull'assicurare che un sistema sia continuamente conforme ai requisiti e alle esigenze funzionali e non-funzionali. Il suo obiettivo primario è mantenere l'affidabilità del servizio bilanciando la velocità di release con la stabilità, tipicamente attraverso automazione, monitoraggio e pianificazione proattiva della capacità. Creando cicli di feedback automatizzati e usando il feedback per apportare miglioramenti, l'SRE supporta il team nell'assicurare che i sistemi restino entro i livelli di servizio definiti, minimizzando i disservizi e ottimizzando l'efficienza, colmando il divario tra sviluppo e operations negli ambienti di produzione.

È prassi comune che un cliente e un fornitore definiscano i livelli di servizio richiesti per garantire le prestazioni operative di un sistema software. Un livello di servizio descrive uno specifico aspetto di un sistema software rilevante per gli stakeholder. Le parti coinvolte formalizzano i propri accordi in Service Level Agreement (SLA). Gli SLA sono impegni vincolanti che aiutano a stabilire le priorità delle attività di sviluppo e di operations e supportano la progettazione di test case pertinenti, assicurando che il servizio sia costantemente allineato agli obiettivi di business e alle esigenze degli utenti.

I Service Level Objective (SLO) e i Service Level Indicator (SLI) sono concetti importanti nell'SRE. Sono necessari per rispettare tali SLA. Gli SLO descrivono a che cosa un sistema software deve essere conforme, mentre gli SLI sono usati per misurare se gli SLO vengono raggiunti. Uno SLO include il valore obiettivo o l'intervallo di valori per un livello di servizio.

Uno SLI è una misura quantitativa ben definita degli aspetti del livello di servizio erogati. Uno o più SLI sono usati per determinare se gli SLO vengono raggiunti, assicurando che il servizio sia erogato al livello concordato. Gli SLI sono misurati durante le fasi di sviluppo attraverso il testing. Durante il processo di operations gli SLI sono misurati tramite il monitoraggio (per esempio con observability e telemetria). L'esecuzione automatizzata dei regression test è importante per misurare gli SLI nei vari stadi della pipeline di CI/CD.

La creazione e l'implementazione di sistemi e servizi nell'SRE seguono tre fasi distinte:

- pianificazione e progettazione dell'infrastruttura necessaria: riguarda caratteristiche di qualità quali affidabilità, scalabilità e sicurezza;

Quality in DevOps (CT-QDO) Syllabus

- implementazione e deployment dell'infrastruttura: comprende la configurazione dell'infrastruttura, il deployment del sistema o del servizio e la verifica che il sistema funzioni secondo il livello di servizio concordato;
- esercizio, manutenzione e ottimizzazione continui di un sistema o di un servizio per mantenerlo affidabile ed efficiente nel tempo.

1.2.3 Pattern e Anti-pattern dei Team DevOps

Per raggiungere l'obiettivo primario di consegnare valore al cliente, organizzazioni diverse possono adottare strutture di team DevOps differenti.

La struttura dei team dell'organizzazione dipende da:

- la complessità del software, poiché una complessità maggiore favorisce i silos;
- la leadership tecnica e la sua capacità di creare un obiettivo condiviso per i team Dev e Ops;
- la disponibilità del reparto IT operations a passare a un modello DevOps;
- le competenze necessarie a guidare la trasformazione verso il modello DevOps.

Un'adeguata integrazione di DevOps nelle topologie di team (Matthew Skelton, 2023) agevolerà la collaborazione e la consegna di valore, mentre gli anti-pattern possono ostacolare questi obiettivi mettendo in evidenza problemi organizzativi.

Topologie dei team DevOps

Di seguito sono discussi tre modelli. Sono indipendenti e non elencati in ordine di preferenza.

- **Collaborazione Dev e Ops:**
vi è una cooperazione fluida tra i team Dev e Ops. Entrambi i team apprendono le basi del modo di lavorare dell'altro. Ciò richiede un obiettivo condiviso chiaramente definito e un significativo cambiamento culturale dell'organizzazione.
- **Responsabilità operative pienamente condivise:** le persone delle operations sono pienamente integrate nei team Dev. Organizzazioni come Facebook e Netflix hanno reso popolare questo modello nei primi anni 2010. In questo modello tutti sono concentrati sullo stesso scopo. I team che costruiscono una funzionalità sono anche responsabili di mantenerne l'esercizio.
- **Team SRE:**
organizzazioni come Google definiscono la collaborazione tra i team Dev e Ops come il team SRE che collabora con i team Dev. Per una spiegazione dettagliata dei team SRE si rimanda alla *Sezione 1.2.2*. I team Dev devono fornire evidenze di test che attestino che i loro prodotti di lavoro soddisfano gli standard fissati dal team SRE, senza essere direttamente coinvolti negli aspetti operativi. Questo pattern può facilmente scivolare in anti-pattern e silos.

Anti-pattern dei team DevOps

- **Silos Dev e Ops:**
questa struttura di team ha una mentalità «al di là del muro». «Done» è tipicamente definito come funzionalità completata e non rilasciata in produzione.

Quality in DevOps (CT-QDO) Syllabus

- Dev non ha bisogno di Ops:
i nuovi team Dev che operano in ambienti che utilizzano servizi cloud potrebbero pensare di non aver bisogno delle Ops.
- DevOps come team di strumenti:
un team DevOps separato implementa gli strumenti necessari per DevOps. Ciò può giovare alla dotazione di strumenti del reparto Dev, ma non raggiunge l'impatto che potrebbero avere veri team DevOps, per via della mancata integrazione nelle pratiche di lavoro quotidiane dei team di sviluppo, dove risiedono molti dei benefici dei modi di lavorare DevOps.

2 Quality Assurance (QA) e Testing in DevOps - 360 minuti

Parole chiave

testing A/B, Acceptance Test-Driven Development, sviluppo software Agile, testing automatizzato, Behavior-Driven Development, canary release, continuous integration, continuous testing, testing olistico, quality engineering, politica della qualità, modello di sviluppo sequenziale, analisi statica, testing statico, politica di test

Parole chiave specifiche di Quality in DevOps

blue-green deployment, pipeline di CI/CD, continuous delivery, continuous deployment, continuous discovery, continuous learning and experimentation, continuous monitoring, dark launch, pull request, release on demand, value stream

Obiettivi di Apprendimento del Capitolo 2:

2.1 Contribuire a Tutte le Fasi del Value Stream a Supporto del Quality Engineering

- QDO-2.1.1 (K3) Implementare attività di Quality Assurance in un contesto DevOps
- QDO-2.1.2 (K2) Confrontare gli obiettivi del test che supportano DevOps con i modelli di sviluppo sequenziali e con lo sviluppo software Agile
- QDO-2.1.3 (K2) Spiegare il continuous testing in un contesto DevOps
- QDO-2.1.4 (K2) Spiegare come le pull request supportano la qualità

2.2 Contribuire all'Implementazione del Loop DevOps

- QDO-2.2.1 (K2) Spiegare Quality Assurance e testing nella continuous discovery e le relative pratiche
- QDO-2.2.2 (K2) Spiegare Quality Assurance e testing nella continuous integration e le relative pratiche
- QDO-2.2.3 (K2) Spiegare Quality Assurance e testing nella continuous delivery e le relative pratiche
- QDO-2.2.4 (K2) Spiegare Quality Assurance e testing nel continuous deployment e le relative pratiche
- QDO-2.2.5 (K2) Spiegare Quality Assurance e testing nel release on demand e le relative pratiche
- QDO-2.2.6 (K3) Applicare le conoscenze di Quality Assurance e testing per implementare una pipeline di CI/CD

Obiettivi Pratici:

2.1 Contributo al Value Stream a Supporto del Quality Engineering (QE)

- QDO-2.1 (H2) Contribuire al Value Stream a Supporto del Quality Engineering

2.2 Implementazione del Loop DevOps

- QDO-2.2 (H2) Contribuire all'Implementazione del Loop DevOps

2.1 Contributo al Value Stream a Supporto del Quality Engineering (QE)

QA e testing sono elementi chiave nel contesto DevOps, lungo tutto il value stream. Un value stream è una serie di processi, attività e workflow interconnessi che consegnano valore ai clienti attraverso un prodotto, un servizio o un'esperienza. Sebbene QA e testing siano svolti in modo diverso nello sviluppo software Agile e nei modelli di sviluppo sequenziali, entrambi hanno un forte impatto sulla qualità del software. In una prospettiva d'insieme, per raggiungere il giusto livello di qualità serve qualcosa in più di QA e testing. Il QE comprende e sovrasta QA e testing: riguarda tutte le attività che contribuiscono a creare il software di cui il cliente ha bisogno.

Questa sezione approfondisce le attività e le pratiche del contesto DevOps, in particolare con un esercizio pratico. Gli obiettivi del QE in DevOps sono confrontati con lo sviluppo software Agile e con i modelli di sviluppo sequenziali.

La sezione finale descrive l'importanza del processo di pull request (PR).

Obiettivo Pratico QDO-2.1 (H2) Contribuire al Value Stream a Supporto del Quality Engineering

Per questo obiettivo pratico l'esercizio di livello H2 supporta i discenti nell'apprendere come creare un quadro d'insieme delle attività di QE specifiche di DevOps per un caso concreto. Ciò comprende quali deliverable di QA e testing debbano essere definiti, con l'obiettivo di raggiungere la giusta qualità per i tre aspetti della qualità: prodotti, processi e persone.

2.1.1 QA e Testing in un Contesto DevOps

Chi pratica DevOps si sforza di incorporare la qualità fin dall'inizio e di implementare un approccio «right first time». Ciò favorisce un modo di lavorare efficiente lungo tutto l'SDLC e realizza efficacemente le esigenze degli stakeholder. In questo senso DevOps è un'ulteriore elaborazione della mentalità Agile. Per realizzare tale elaborazione, qualsiasi team DevOps o gruppo di team DevOps che collaborano deve implementare attività finalizzate a ottenere la built-in quality e a fornire informazioni sul livello di qualità raggiunto. Una politica della qualità e una politica di test documentate supporteranno questo obiettivo. Gli elenchi seguenti mostrano un raggruppamento di tali principi, attività e pratiche per i tre aspetti della qualità: prodotti, processi e persone (Marselis et al., 2020):

Qualità dei Prodotti

La qualità del prodotto creata dal team o dai team DevOps dipende, per esempio, dai seguenti principi:

- il conseguimento del valore di business è l'obiettivo principale dello sviluppo;
- implementare un'architettura che abiliti la produttività del team DevOps;
- implementare correttamente le caratteristiche di qualità funzionali e non-funzionali del software (Kim; Humble, et al., 2016; *Systems and software engineering (25010)*, 2023);
- prestare attenzione alle attività di quality engineering che devono essere applicate durante tutte le attività del ciclo di vita DevOps; ciò comprende attività di qualità preventive, quali la scrittura collaborativa delle user story e il pair programming, e attività di qualità rilevative, quali il testing statico e il testing dinamico.

Qualità dei Processi

I processi che mirano a costruire il giusto livello di qualità comprendono, per esempio, le seguenti attività:

- fornire monitoraggio, controllo, reporting e alerting pienamente integrati in relazione agli indicatori misurati per il valore di business e il livello di qualità;
- applicare un approccio basato sul rischio per stimare gli sforzi e schedulare le attività e le lavorazioni successive;
- predisporre l'infrastruttura e gli strumenti necessari per le attività di QE integrate in una pipeline di CI/CD;
- implementare pratiche di miglioramento continuo.

Qualità delle Persone

Le persone coinvolte devono applicare, come parte della mentalità Agile, per esempio le seguenti pratiche per consegnare correttamente i prodotti:

- sviluppo guidato dalle ipotesi: (riteniamo che <una certa azione> produrrà <un obiettivo atteso>. Avremo la fiducia necessaria per proseguire quando <una misurazione supererà un certo valore obiettivo>) (Kim; Humble, et al., 2016);
- un ambiente di lavoro collaborativo;
- un'attenzione al miglioramento continuo.

Per implementare questi principi, attività e pratiche un'organizzazione deve predisporre linee guida di riferimento per tutti i team DevOps e per gli stakeholder coinvolti. I prodotti di lavoro principali sono la politica della qualità e la politica di test, che possono essere unite in un unico documento, con un approccio test-first per la built-in quality e una mentalità «automatizza tutto il possibile». La politica della qualità e la politica di test descrivono i ruoli e le responsabilità dei team cross-funzionali, descrivono l'organizzazione di più team e ne delineano l'implementazione. L'implementazione delle politiche può basarsi su un'analisi del rischio di prodotto, sulla piramide di test e/o sui quadranti del testing Agile (*ISTQB® CTFL*, v4.0) e produrre una strategia di QE (Marselis et al., 2020). L'implementazione richiede la collaborazione dei membri del team DevOps e tra più team DevOps.

2.1.2 Confrontare gli Obiettivi del Test nei Diversi Modelli di SDLC

I tre modi di DevOps (cioè flusso, feedback e apprendimento continuo) (Kim; Humble, et al., 2016) richiedono un approccio pienamente integrato a QA e testing. La qualità deve essere incorporata fin dall'inizio e il testing fornisce informazioni sul fatto che il giusto livello di qualità sia consegnato nel momento giusto.

Nei modelli di sviluppo sequenziali il testing viene svolto in livelli di test separati ed è effettuato da team di test specializzati.

Nello sviluppo software Agile il team Agile esegue il testing per confermare che una user story sia completata e che i criteri di accettazione siano soddisfatti. Rinviare le attività di test alla fine di un'iterazione può portare a inefficienze e a rischi per la qualità.

DevOps estende la mentalità Agile: il principio del flusso stabilisce che ogni prodotto di lavoro dovrebbe essere completato e consegnato in modo indipendente, e il testing è parte integrante dello sviluppo.

Gli obiettivi del test sono simili in tutti e tre gli approcci citati, in quanto forniscono informazioni sulla qualità per costruire fiducia, mentre gli obiettivi del test nello sviluppo software Agile e in DevOps comprendono anche l'esigenza di fornire feedback precoce e continuo.

In DevOps l'esecuzione automatizzata dei test è essenziale in una pipeline di CI/CD. La test automation è essenziale per realizzare il flusso. A differenza dei modelli di sviluppo sequenziali, dove i team possono avere successo senza alcuna esecuzione automatizzata dei test, e dello sviluppo software Agile, dove il testing automatizzato è spesso concentrato sul regression testing, in DevOps la test automation si applica a tutti i livelli di test e a tutti i tipi di test. Tuttavia, anche in DevOps, pur utilizzando pipeline di CI/CD automatizzate, resterà comunque una certa necessità di testing manuale, in particolare di testing esplorativo. Il testing manuale è necessario perché alcuni aspetti della qualità sono difficili da valutare in modo automatizzato (per esempio l'usabilità). Infine, il testing manuale è adatto agli stakeholder che vogliono vedere i risultati dei test di persona prima di convincersi che il livello di qualità del software sia adeguato.

2.1.3 Continuous Testing

Il continuous testing integra le attività di test lungo tutto il ciclo di vita dello sviluppo software (SDLC), cosicché il testing avviene dal concetto iniziale fino al deployment in produzione e oltre, anziché costituire una fase separata.

Principi fondamentali del continuous testing

- **Testare precocemente e spesso**
Il continuous testing pone l'accento sull'avviare le attività di test il prima possibile e sul mantenerle con continuità lungo tutto l'SDLC. Verificando e validando il software in ogni fase, i team DevOps valutano e riportano il livello di qualità precocemente, riducendo in misura significativa il costo e l'impatto dei difetti.
- **Collaborazione cross-funzionale**
Nel continuous testing la qualità è responsabilità di tutti. L'approccio richiede la partecipazione attiva di tutti i membri del team DevOps, non solo dei tester dedicati. Questo approccio collaborativo assicura prospettive diversificate e valutazioni della qualità complete. È analogo all'«approccio whole team» (*ISTQB® CT-TAS*, v1.0).
- **Automazione e integrazione degli strumenti**
Per rendere efficiente il continuous testing i team DevOps si affidano in misura rilevante all'automazione. I test automatizzati nella pipeline di CI/CD forniscono feedback rapido e consentono una qualità consistente. Si utilizzano strumenti per i vari tipi di test per ottenere una copertura completa.
- **Monitoraggio in produzione e observability**
Il continuous testing si estende oltre gli ambienti di pre-produzione, fino alla produzione. Il continuous testing sottolinea l'importanza del monitoraggio e della observability negli ambienti di produzione. I team DevOps utilizzano strumenti specializzati per tracciare le prestazioni del software in produzione, rilevare anomalie e raccogliere dati sul comportamento degli utenti in tempo reale, estendendo così la fase di test alla produzione. Per maggiori informazioni su monitoraggio e telemetria si veda la *Sezione 4.2.7*.
- **Miglioramento adattivo**
Il continuous testing promuove un affinamento continuo delle strategie di test. I team DevOps

raccogliono con continuità dati e feedback dalle varie attività di test, utilizzando queste informazioni per adattare le proprie strategie di test e per migliorare nel tempo i propri approcci del test.

Esempio di framework

Un framework che esemplifica il continuous testing è il testing olistico sviluppato da Janet Gregory e Lisa Crispin (Janet Gregory, 2023).

Questo framework aiuta i team DevOps a strutturare e applicare il continuous testing in scenari di sviluppo reali, semplificandone l'adozione e l'adattamento alle proprie esigenze.

2.1.4 Pull Request

Una PR, talvolta denominata merge request, è un accordo che richiede all'autore del codice di richiedere la review del codice da parte di uno o più membri del team prima che il codice possa essere integrato nel ramo principale. In una PR un ramo è una linea di sviluppo separata in un version control system (VCS), dove i cambiamenti avvengono prima di essere integrati nella codebase principale.

Il concetto di PR è nato dalle pratiche dei VCS. Prima che l'archiviazione in cloud diventasse lo standard di fatto per i VCS, gli sviluppatori scrivevano il nuovo codice sulla propria macchina locale e gli altri sviluppatori lo «prelevavano» (pull) per contribuire a valutarne la qualità.

Il loro feedback viene incorporato nel prodotto di lavoro e supporta una qualità più elevata in diversi modi.

- Le PR supportano la qualità in DevOps incoraggiando uno sviluppo collaborativo e la condivisione della conoscenza tra i membri del team, con conseguenti migliori pratiche di codifica e una maggiore consapevolezza dei problemi potenziali.
- Le PR possono fungere da quality gate per avviare un'esecuzione automatizzata dei test nelle pipeline di CI/CD, come discusso nella sezione 2.2.
- Le PR contribuiscono a far rispettare gli standard di codifica, l'aderenza alle guide di stile e le best practice attraverso review manuali del codice supportate dall'analisi statica eseguita nella pipeline di CI/CD.
- Le PR migliorano la tracciabilità fornendo una storia chiara dei cambiamenti, delle discussioni e delle decisioni prese su specifiche funzionalità o correzioni, favorendo la responsabilità.
- Incoraggiando cambiamenti di codice piccoli e gestibili anziché aggiornamenti ampi e complessi, le PR limitano l'entità del cambiamento e i rischi associati.
- Agendo da quality gate, le PR assicurano che solo i cambiamenti approvati confluiscono nel ramo principale.
- Le PR promuovono la responsabilità facendo in modo che specifici membri del team DevOps esaminino e approvino i cambiamenti, supportando una condivisione intenzionale della conoscenza, comprese le review da parte di tutti i team interessati dai cambiamenti di codice.

2.2 Implementazione del Loop DevOps

Un value stream è una serie di attività che vanno dalla raccolta di un'idea fino al rilascio di valore ai clienti. Sviluppo, consegna ed esercizio del software non sono insiemi di attività distinti che avvengono in isolamento: al contrario, si svolgono in stretta prossimità, in particolare con i modi di lavorare dello sviluppo software Agile e di DevOps (DASA, 2019).

Un value stream è illustrato come un loop a forma di infinito, il cosiddetto loop DevOps. Sebbene esistano diverse rappresentazioni del loop DevOps con sequenze di attività differenti, il gruppo principale di attività è il medesimo. Questa sezione tratta il modo in cui QE, QA e testing sono considerati nei gruppi di attività del value stream, in particolare in continuous discovery, continuous integration (CI), continuous delivery (CD) e continuous deployment, release on demand e continuous monitoring.

Il QE, che comprende QA e testing, avviene in tutte le attività del loop DevOps, che contribuiscono tutte alla qualità del software. Queste attività possono procedere rapidamente o lentamente, a seconda del numero di elementi noti o ignoti, e talvolta un team DevOps dovrà fermarsi e tornare a una o due attività precedenti per ripercorrerle prima di procedere nuovamente (per esempio per capire se una funzionalità ha senso) (Janet Gregory, 2023).

Sebbene nel value stream possano esservi passi di processo manuali, in DevOps è consueto puntare all'automazione.

La pipeline di CI/CD si riferisce all'automazione relativa al codice, compreso il reporting sull'esito delle attività. Tuttavia non comprende tutte le attività del value stream. Altri strumenti possono supportare tali attività, come gli strumenti di gestione delle attività, dove è possibile automatizzare i task. Gli strumenti di gestione delle attività e quelli CI/CD sono spesso strettamente collegati.

Le sezioni 2.2.1-2.2.5 descrivono le attività del loop DevOps. La sezione 2.2.6 descrive i passi per implementare incrementalmente le pipeline di CI/CD nell'organizzazione a copertura di tali attività.

Obiettivo Pratico QDO-2.2 (H2) Contribuire all'Implementazione del Loop DevOps

Per questo obiettivo pratico l'esercizio di livello H2 supporta i discenti nell'apprendere come progettare e implementare una pipeline di CI/CD, quali stadi prevedere e quali capability degli strumenti includere per rappresentare un loop DevOps.

2.2.1 QA e Testing nella Continuous Discovery

La continuous discovery è una pratica consistente nello scoprire e valutare frequentemente scoperte, esigenze dei clienti e degli stakeholder, idee innovative, tecnologie moderne e altre opportunità, per decidere quale valore sviluppare successivamente. I team DevOps si concentrano sulla costruzione di funzionalità che generano valore coinvolgendo continuamente gli stakeholder, applicando la centralità del cliente e il design thinking e scoprendone le esigenze (Torres, 2024), (*Scaled Agile Framework*, 2023b), (*Scaled Agile Framework*, 2023c).

Di norma nella continuous discovery si svolgono tre attività: discover, plan e understand (Janet Gregory, 2023).

- Discover riguarda l'aggiunta di nuove funzionalità e cambiamenti al software affinché aiuti gli stakeholder a risolvere i loro problemi e a soddisfare le loro esigenze. Gli stakeholder validano

Quality in DevOps (CT-QDO) Syllabus

tali esigenze e idee prima che le funzionalità vengano definite. Ciò può essere realizzato, per esempio, attraverso interviste agli stakeholder o sviluppando rapidi prototipi di user experience (UX). I product owner, supportati dal team DevOps, individuano le funzionalità che consentiranno al team di creare, migliorare e consegnare le soluzioni per gli stakeholder.

- Plan determina le caratteristiche di qualità e individua i rischi rilevanti per la funzionalità, dettagliando al contempo le funzionalità e scomponendole in user story più piccole. Poiché DevOps utilizza un approccio iterativo/incrementale, gli elementi del backlog sono individuati e dettagliati in modo incrementale, sulla base di cicli di apprendimento brevi, anziché definire tutto in anticipo.
- Understand riguarda l'ottenimento di una visione condivisa dei dettagli delle user story all'interno del team DevOps o tra i diversi team coinvolti.
I membri del team DevOps applicano un approccio test-first, utilizzando l'Acceptance Test-Driven Development (ATDD) o il Behavior-Driven Development (BDD), che comprende la descrizione della specifica per esempi (Adzic, 2011). Questi scenari di esempio aiutano gli sviluppatori a comprendere e testare la soluzione. Questo approccio test-first è un'attività di test collaborativa che supporta la built-in quality. La comprensione dell'elemento di backlog da parte dei membri del team DevOps è garantita soddisfacendo la definition of ready (DoR) (per esempio definire e comprendere i criteri di accettazione e stimare gli elementi del backlog) (Alliance, 2026). Una definition of done (DoD) condivisa guida l'individuazione di tutti i prodotti di lavoro da consegnare per ciascun elemento del backlog (Alliance, 2026).

Una best practice di QA consiste nell'avere checklist di DoR e DoD predisposte, definite, concordate e seguite dai membri del team DevOps.

2.2.2 QA e Testing nella CI

Per ogni cambiamento integrato, la CI richiede di costruire l'intero software e di eseguire su di esso un insieme completo di test automatizzati, sia statici sia dinamici (Beck, 2000). Violazioni dello stile di codifica, mancato raggiungimento degli obiettivi di copertura o cicli di test lenti possono determinare una build interrotta. Se la build o i test falliscono, i membri del team interrompono tutto ciò che stanno facendo e correggono immediatamente la build. Questo approccio «stop-and-fix» è un aspetto di quality management del lean, come discusso in (Humble; Molesky, et al., 2020). La correzione può consistere nel tornare alla precedente versione funzionante del software.

L'obiettivo della CI è avere il software sempre in uno stato funzionante, così da renderlo potenzialmente rilasciabile. «Potenzialmente rilasciabile» è un'affermazione sulla qualità del software e non sul suo valore o sulla sua commerciabilità (Larman et al., 2016).

I team DevOps devono disporre delle seguenti pratiche per implementare la CI, come descritto in (Humble; Farley, 2010):

- includere tutto nella gestione della configurazione (per esempio script di test automatizzati, script di deployment automatizzati, dati di test e configurazioni degli ambienti);
- gli script di automazione dei passi di build o di test nella pipeline di CI/CD dovrebbero agevolare l'indagine in caso di failure della build;
- i team DevOps effettuano il check-in di cambiamenti piccoli e incrementali e attendono che i test passino, oppure seguono un approccio stop-and-fix se un test fallisce.

Quality in DevOps (CT-QDO) Syllabus

Come descritto in (Humble; Farley, 2010), la pipeline di CI si compone di norma di due parti principali:

- Le azioni nel commit stage verificano che il software funzioni a livello tecnico. Questo attiva compilazioni, esecuzione automatizzata di test di basso livello e analisi statica del codice. La review del codice da parte degli altri membri del team DevOps avviene di norma solo dopo che le azioni del commit stage sono state superate.
- Le azioni nell'acceptance stage automatizzato verificano che il software funzioni, sia conforme alla specifica e soddisfi le esigenze dell'utente. L'acceptance stage può comprendere qualsiasi tipo di test o livello di test necessario per accettare il software. In questo stadio le attività di test possono essere eseguite in parallelo.

Quando più team DevOps contribuiscono al software, di norma ciascun team ha la propria parte di CI specifica del team all'interno della pipeline di CI/CD, collegata a una parte di CD comune e a livello organizzativo della pipeline di CI/CD (vedi *Sezione 2.2.3* e *Sezione 2.2.4*).

2.2.3 QA e Testing nella CD

La continuous delivery (CD) assicura che il software sia sempre in uno stato rilasciabile, anche verso gli ambienti di produzione. Se il software venga effettivamente rilasciato in produzione è una decisione che spetta a una persona, sulla base dei risultati dei test e delle valutazioni di prontezza (a differenza del continuous deployment, dove il deployment avviene automaticamente se tutti i test sono superati, senza alcun intervento umano, come descritto nella *Sezione 2.2.4*).

Dopo gli stadi di CI (vedi *Sezione 2.2.2*) il software viene rilasciato su uno o più ambienti di test simili alla produzione per ulteriori test. Tali test possono utilizzare la test automation, ricorrere a tecniche di test basate sull'esperienza o entrambe. È prassi comune eseguire smoke test dopo il deployment su un ambiente e prima dell'esecuzione dei test pianificati, per verificare che il deployment sia andato a buon fine. I test possono essere eseguiti su più ambienti in parallelo per fornire feedback rapido, se le risorse necessarie sono disponibili.

La qualità si ottiene attraverso il testing e mediante le seguenti pratiche di QE:

- automazione del deployment tramite strumenti di automazione, per consistenza, conformità e per fornire un audit trail (vedi *Sezione 4.1*);
- gestione standardizzata degli ambienti di test e dei dati di test (vedi *Sezione 3.1.6*);
- IaC per gestire la configurazione dell'infrastruttura tramite codice, per consistenza e ripetibilità (vedi *Sezione 4.2.3*).

La strategia di rilascio adottata dal team DevOps (vedi *Sezione 4.2.2*) fornisce linee guida su quanto spesso e come effettuare il deployment in produzione (vedi *Sezione 2.2.4*) e su come rilasciare agli utenti (vedi *Sezione 2.2.5*).

2.2.4 QA e Testing nel Continuous Deployment

Il continuous deployment si fonda sulla pipeline di CI/CD e sulle pratiche di CI e CD. Il continuous deployment differisce dalla continuous delivery (CD) perché rilascia automaticamente in produzione ogni cambiamento che ha superato i propri test automatizzati. Non vi è coinvolgimento umano nelle decisioni di deployment. Questo approccio assicura che il software sia sempre in uno stato rilasciabile e che la versione più recente sia rilasciata in produzione.

Quality in DevOps (CT-QDO) Syllabus

Il deployment in produzione non significa necessariamente che il software sia rilasciato agli utenti. Per i dettagli si veda la *Sezione 2.2.5*. Tuttavia una procedura di deployment ripetuta frequentemente e affidabile riduce il rischio di un deployment fallito durante il release del software.

La fiducia necessaria per effettuare sempre il deployment in produzione si fonda sui seguenti fattori:

- disporre di pratiche di QE che incorporino la qualità lungo continuous discovery, CI e CD;
- disporre di un processo di deployment automatizzato e controllato, senza coinvolgimento umano;
- avere fiducia nella test suite automatizzata della pipeline di CI/CD per raggiungere livelli di qualità concordati e accettabili;
- applicare strategie di deployment e di release per controllare come rilasciare l'applicazione e come fornire l'accesso agli utenti (per esempio il dark launch) (vedi *Sezione 4.2.2*);
- avere la capacità di rispondere rapidamente alle conseguenze indesiderate del deployment (per esempio effettuare il rollback a uno stato del sistema definito in precedenza oppure il roll forward rilasciando una versione più recente del sistema per correggere i difetti).

Il testing avviene in produzione prima del rilascio del software agli utenti, per ridurre ulteriormente il rischio e generare feedback sulla nuova funzionalità o su altre caratteristiche di qualità (vedi *Sezione 3.3.3*).

- Con un dark launch la nuova funzionalità viene eseguita nell'ambiente di produzione accanto alle funzionalità esistenti senza essere esposta agli utenti, consentendo ai team DevOps di monitorare prestazioni e comportamento, testare e osservare il sistema in condizioni reali.
- Nel blue-green deployment si utilizzano due ambienti identici, «blue» (quello attualmente in esercizio) e «green» (la nuova versione), per minimizzare i tempi di fermo e il rischio durante il deployment. Il testing avviene sulla nuova versione (green) mentre la vecchia versione (blue) resta in esercizio. Dopo che tutti i test sono stati superati, l'ambiente green va in esercizio e l'ambiente blue è pronto per il cambiamento successivo e per i relativi test: i due si scambiano quindi i ruoli. Due ambienti consentono un rollback rapido se emergono problemi dopo il rilascio di una nuova versione.
- Una canary release espone dapprima un piccolo sottoinsieme di utenti a un cambiamento software per validarne la qualità, per poi estenderlo incrementalmente all'intera base utenti se non vi sono difetti critici. Il primo sottoinsieme di utenti può essere una percentuale casuale della base utenti totale, un gruppo preselezionato di utenti noti (cioè beta tester), un'area geografica o qualsiasi altro raggruppamento.
- Il testing A/B è un approccio del test per confrontare due o più varianti di una funzionalità, di un'interfaccia o di una funzione presentando ciascuna variante a segmenti differenti di utenti e analizzando le metriche di prestazione per determinare quale versione consegua più efficacemente il risultato desiderato, mantenendo quella variante in produzione e rimuovendo le altre.

2.2.5 QA e Testing nel Release on Demand

Il release on demand è una pratica DevOps che consiste nel rilasciare nuove funzionalità immediatamente o in modo incrementale sulla base delle esigenze del business, dei clienti e di ogni altro stakeholder, disaccoppiando deployment e release. Quando una nuova funzionalità viene rilasciata in

produzione, resta nascosta agli utenti fino a quando l'accesso non viene fornito in modo controllato mediante pratiche e strumenti di feature management (vedi *Sezione 4.2.2* e *Sezione 4.2.4*). Questo approccio consente al business di rilasciare le funzionalità agli utenti nel momento di mercato migliore (per esempio a una data prestabilita prima del periodo delle festività). Supporta la business agility, consegnando e mantenendo il massimo valore per gli utenti quando ne hanno bisogno e per tutto il tempo in cui ne hanno bisogno.

Disaccoppiare gli elementi architetturali affinché siano rilasciabili separatamente riduce il rischio di interruzioni del servizio durante deployment e release. Questa tecnica individua specifici building block architetturali, ciascuno dei quali può essere rilasciato in modo indipendente (per esempio le basi di dati di backend open-source seguono cicli di release maggiori, mentre i microservizi di front-end sono aggiornati più volte per iterazione). I diversi building block possono avere strategie e frequenze di release differenti.

Il release può comportare attività non legate al software (per esempio l'aggiornamento di manuali utente, materiali di formazione, note di rilascio, procedure operative, contratti legali oppure attività di marketing e vendita).

Dopo il rilascio del software le operations si concentrano su attività critiche che ne preservano la stabilità e la sicurezza (vedi *Sezione 1.2.2*). Le pratiche di continuous monitoring sono utilizzate per osservare il software nel suo ambiente, per fornire informazioni sul fatto che la qualità del software soddisfi gli obiettivi concordati e per poter intraprendere azioni correttive dopo un alert su un degrado della qualità (vedi *Sezione 4.2.7*). Anche le attività di IT service management (ITSM) supportano la qualità (per esempio l'istituzione di un service desk che fornisca supporto a clienti e utenti).

L'observability misura il comportamento del sistema rispetto ai suoi requisiti funzionali e non-funzionali (vedi *Sezione 4.2.7*). Inoltre si raccolgono informazioni da altri canali (per esempio il feedback dei clienti dal service desk o dai social media). I dati raccolti aiutano a capire come il cambiamento rilasciato produca benefici di business (per esempio un maggior numero di sottoscrizioni o un miglioramento delle vendite).

L'organizzazione può utilizzare metriche per applicare quanto appreso e individuare il valore potenziale successivo da consegnare. Questo può essere un miglioramento dell'ultima funzionalità, un miglioramento non-funzionale o l'avvio di una nuova funzionalità. Inoltre l'organizzazione può applicare quanto appreso per migliorare processi, pratiche, persone e architettura del prodotto.

2.2.6 Implementazione della Pipeline di CI/CD

L'implementazione di una pipeline di CI/CD dovrebbe seguire un approccio incrementale con i passi seguenti, così come definiti in (Humble; Farley, 2010).

1. Modellare il value stream dal commit al release e creare un walking skeleton

La tecnica del value stream mapping (VSM) mappa la parte del value stream che va dal check-in del codice al release (per i dettagli sul VSM vedi (*ISTQB® CT-ATLaS, v2.0*)). I passi mappati vengono poi implementati senza contenuto dettagliato nella pipeline di CI/CD. Quando i quality gate predefiniti sono soddisfatti, ciò attiva il passo successivo del processo mappato e l'output di un passo diventa input per gli altri (per esempio artefatti di build e basi di dati di test).

Il codice della pipeline di CI/CD è di norma memorizzato nello stesso repository del codice sorgente dell'applicazione. Tuttavia, nei progetti più complessi, i workflow possono essere riutilizzati tra più repository.

Un walking skeleton mostra il flusso e collega i passi prima di aggiungere i dettagli.

2. Automatizzare il processo di build e di deployment

Il toolset di CI dovrebbe attivare una build a ogni check-in. Il processo di build prende come input il codice sorgente e produce come output i programmi. Dopo la build l'output viene rilasciato automaticamente su un ambiente di test e i test vengono eseguiti.

Il provisioning di un ambiente e dei dati di test dovrebbe essere completamente automatizzato e riproducibile. Le tecnologie di IaC, virtualizzazione e containerizzazione supportano questo passo (vedi rispettivamente *Sezione 4.2.3* e *Sezione 4.2.9*). I ruoli di operations sono di norma coinvolti nello sviluppo di questa automazione (vedi *Sezione 1.2.1* e *Sezione 1.2.2*). Gli ambienti di test sono rilasciati nello stesso modo degli ambienti di produzione.

3. Automatizzare il commit stage

Il passo successivo consiste nell'implementare un commit stage completo (vedi *Sezione 2.2.2*), eseguendo analisi statica e unit test e includendo una selezione di test di componente, test di integrazione e test di sistema a ogni commit. In questo stadio lo scanning delle vulnerabilità e l'analisi delle dipendenze implementano le pratiche di sicurezza. È prassi comune implementare un quality gate per verificare se la build debba essere interrotta qualora le soglie concordate non siano rispettate (per esempio copertura del codice insufficiente o tempo di esecuzione dei test troppo lungo). Quando il commit stage diventa troppo lungo (per esempio oltre cinque minuti), è sensato suddividerlo in job paralleli.

4. Automatizzare l'acceptance stage

L'implementazione dell'acceptance stage (vedi *Sezione 2.2.2*) dovrebbe iniziare automatizzando alcuni test case di tutti i diversi tipi di test nella pipeline di CI/CD e facendo crescere le test suite in modo incrementale, invece di cercare di automatizzare la maggior parte dei test case di un tipo di test prima di passare al successivo.

I test vengono eseguiti in sequenza o in parallelo, ciascuno nel proprio ambiente, per ottimizzare il feedback. Occorre però considerare la disponibilità delle risorse, i costi e gli obiettivi di sostenibilità.

5. Automatizzare il release

Automatizzare questo processo richiede di comprendere il modo in cui la decisione di release viene presa durante il processo di release (vedi *Sezione 4.2.2*) e richiede di raccogliere le informazioni necessarie a prendere tale decisione, sia essa automatizzata o manuale. Gli strumenti di feature management possono supportare i decisori nella gestione delle proprie decisioni (per esempio aggiornando i feature toggle nel codice) (vedi *Sezione 4.2.4*). L'automazione incrementale si applica anche a queste attività di release.

Il processo implementato dovrebbe seguire la strategia di deployment e di release concordata, supportare un rollback rapido e fornire un audit trail di tutte le attività ai fini della conformità.

6. Far evolvere la pipeline di CI/CD

Implementare la pipeline di CI/CD in modo incrementale. Man mano che il progetto diventa più complesso, il value stream evolverà. Misurare l'efficienza della pipeline di CI/CD utilizzando metriche concordate (per esempio il lead time di una qualsiasi attività). Far evolvere e migliorare con continuità la

Quality in DevOps (CT-QDO) Syllabus

pipeline di CI/CD in linea con i principi lean e con il systems thinking (per esempio suddividendo esecuzioni di test lunghe in job paralleli oppure eliminando job che non aggiungono valore). Per le pratiche di miglioramento continuo si vedano (*ISTQB® CT-ATLaS*, v2.0) e (*ISTQB® CTAL-TM*, v3.0).

3 Attività Automatizzate e Manuali in DevOps - 510 minuti

Parole chiave

testing delle API, testing automatizzato, testing di contratti, crowd testing, testing esplorativo, quality hunting, quality report, reporting della qualità, regression testing, test automation, dati di test, gestione dei dati di test, risultato del test, tracciabilità

Parole chiave specifiche di Quality in DevOps

artefatto di build, pipeline di CI/CD, tempo di recupero da un deploy fallito, Personally Identifiable Information, quality report, Single Source of Truth, file binario, analisi statistica

Obiettivi di Apprendimento del Capitolo 3:

3.1 Descrivere Come l'Automazione Supporta la Quality Assurance in DevOps

- QDO-3.1.1 (K2) Spiegare come una Single Source of Truth per il testware supporta la Quality Assurance
- QDO-3.1.2 (K2) Spiegare come l'automazione supporta la tracciabilità tra la base di test e il testware
- QDO-3.1.3 (K3) Implementare pratiche uniformi di reporting della qualità nella pipeline di CI/CD con informazioni provenienti sia dal testing automatizzato sia dal testing manuale
- QDO-3.1.4 (K2) Spiegare i benefici dell'automazione della gestione dei dati di test
- QDO-3.1.5 (K2) Spiegare i benefici dell'analisi statistica dei risultati dei test su lunghi periodi di tempo
- QDO-3.1.6 (K2) Spiegare i benefici di una gestione degli ambienti di test standardizzata, controllata e automatizzata

3.2 Implementare il Testing Automatizzato nei Team DevOps

- QDO-3.2.1 (K2) Spiegare come il regression testing si integra nelle varie attività della pipeline di CI/CD
- QDO-3.2.2 (K3) Predisporre il testing delle API
- QDO-3.2.3 (K2) Confrontare l'estensione della test automation in DevOps rispetto allo sviluppo software Agile e ai modelli di sviluppo sequenziali

3.3 Implementare il Testing Manuale nei Team DevOps

- QDO-3.3.1 (K2) Fornire esempi di testing manuale per un'organizzazione DevOps
- QDO-3.3.2 (K2) Spiegare come il testing esplorativo può funzionare con una pipeline di CI/CD
- QDO-3.3.3 (K2) Spiegare come il crowd testing può supportare la cultura del feedback
- QDO-3.3.4 (K3) Applicare eventi di quality hunting a supporto della cultura dell'apprendimento

Obiettivi Pratici:

3.1 Il Supporto dell'Automazione alla Quality Assurance (QA) in DevOps

- QDO-3.1 (H2) Descrivere Come l'Automazione Supporta la Quality Assurance in DevOps

3.2 Testing Automatizzato nei Team DevOps

QDO-3.2 (H2) Test Automation nei Team DevOps

3.3 Testing Manuale nei Team DevOps

QDO-3.3 (H2) Implementare il Testing Manuale nei Team DevOps

3.1 Il Supporto dell'Automazione alla Quality Assurance (QA) in DevOps

L'automazione svolge un ruolo cruciale nella QA in DevOps migliorando consistenza, tracciabilità ed efficienza lungo tutto l'SDLC. L'implementazione di una Single Source of Truth (SSOT) aiuta a centralizzare e standardizzare le informazioni, assicurando che i team lavorino con informazioni accurate e aggiornate, il che favorisce la collaborazione e riduce gli errori. La tracciabilità automatizzata tra requisiti, testware e deployment assicura la responsabilità su ogni cambiamento, migliorando il defect management e la conformità. Il reporting della qualità integrato con la pipeline di CI/CD fornisce informazioni in tempo reale tramite dashboard, combinando i risultati dei test automatizzati e manuali per un'analisi completa e per l'alerting. Ciò offre a team e stakeholder un feedback rapido e affidabile sulla qualità. L'automazione nella gestione dei dati di test, nell'analisi statistica e nel provisioning degli ambienti di test accelera i cicli di test, contribuisce a garantire la sicurezza dei dati, ottimizza l'uso delle risorse e migliora il processo decisionale, promuovendo il miglioramento continuo della qualità in DevOps.

Obiettivo Pratico QDO-3.1 (H2) Descrivere Come l'Automazione Supporta la Quality Assurance in DevOps

Per questo obiettivo pratico l'esercizio di livello H2 supporta i discenti nell'apprendere come implementare l'automazione a supporto della QA in DevOps. Questo esercizio consente ai discenti di applicare i concetti di automazione a un contesto organizzativo noto e di ragionare su come l'automazione supporti la QA lungo il value stream DevOps.

3.1.1 Single Source of Truth (SSOT) per il Testware

Man mano che le organizzazioni DevOps crescono, gestire molteplici fonti di informazione diventa sempre più complesso. Informazioni frammentate, non consistenti e duplicate tra team DevOps e strumenti diversi conducono a errori, a una collaborazione scadente e a difficoltà nel mantenere trasparenza e tracciabilità tra gli elementi di configurazione. Applicando un'architettura e pratiche di Single Source of Truth (SSOT) a livello organizzativo, i team DevOps possono centralizzare e standardizzare le informazioni, assicurandosi che un dato tipo di informazione sia archiviato in un solo luogo. Ciò garantisce che tutti i team accedano a dati accurati, consistenti e aggiornati, contribuendo a prevenire gli errori dovuti all'affidamento su informazioni errate.

I prodotti di lavoro di sviluppo e di testing vanno archiviati in sistemi SSOT (vedi *Sezione 4.1.1*) come segue:

- i sistemi di gestione delle informazioni forniscono la gestione del ciclo di vita delle informazioni archiviando gli elementi del backlog (per esempio requisiti, defect report e attività) e la documentazione di progetto e di prodotto;
- i version control system (VCS) supportano un change management controllato archiviando il codice sorgente del software, gli script di build e di deployment, le configurazioni di infrastrutture as code (IaC), le definizioni delle pipeline di CI/CD, i test case, il codice di test automation e i dati di test (per esempio dati in formato testuale);
- i sistemi di gestione degli artefatti supportano la riutilizzabilità degli artefatti di build archiviando dipendenze, file binari e dati di test (per esempio dati binari);

Quality in DevOps (CT-QDO) Syllabus

- i sistemi di observability forniscono tracciabilità e gestione degli accessi archiviando tracce, log e metriche.

In generale i sistemi SSOT offrono ai team DevOps i seguenti benefici:

- Centralizzazione: fungono da unico luogo in cui i dati critici sono archiviati, aggiornati e recuperati.
- Consistenza: assicurano che tutti i team lavorino con le stesse informazioni aggiornate, evitando discrepanze.
- Automazione: agevolano workflow automatizzati integrandosi con strumenti e sistemi dell'intera toolchain DevOps.
- Collaborazione: promuovono l'allineamento tra i team fornendo una comprensione condivisa delle informazioni.
- Auditability: forniscono una storia chiara dei cambiamenti, che aiuta a tracciare e garantire la conformità.

3.1.2 Tracciabilità tra la Base di Test e il Testware

Per i team DevOps è importante registrare la relazione tra i prodotti di lavoro correlati, quali requisiti, user story, codice e testware. La tracciabilità assicura che tutti i prodotti di lavoro dell'SDLC siano collegati, consentendo ai team di verificare che ogni requisito sia stato implementato, testato, consegnato, rilasciato in produzione e reso disponibile. La tracciabilità bidirezionale supporta anche la manutenzione (*ISTQB® CTFL*, v4.0). Per esempio, quando viene apportato un cambiamento a un requisito, è facile individuare dove implementarlo. Inoltre la root cause di un difetto del codice può essere individuata rapidamente in una user story o in un requisito. La politica di test dell'organizzazione descrive la necessità della tracciabilità (Marselis et al., 2020).

Gli strumenti possono registrare e monitorare efficientemente la tracciabilità di tutti i prodotti di lavoro correlati. Gli strumenti di automazione sono fondamentali per fornire aggiornamenti dinamici alle matrici di tracciabilità e per ridurre i rischi associati al tracciamento manuale. L'automazione è fornita da strumenti di gestione della configurazione e/o di gestione delle attività, capaci di registrare le varie relazioni tra i prodotti di lavoro. Questi strumenti forniscono report e quadri d'insieme dei prodotti di lavoro disponibili nell'organizzazione e ne dettagliano l'utilizzo.

La tracciabilità favorisce la collaborazione tra i ruoli di sviluppo, testing e operations all'interno del team DevOps rendendo trasparenti e verificabili le relazioni tra i prodotti di lavoro. Questa trasparenza migliora il tracciamento e la risoluzione dei difetti e supporta inoltre la conformità a standard e normative. È un abilitatore chiave di continuous discovery, continuous integration, continuous delivery, continuous deployment, continuous testing e continuous monitoring, e contribuisce a mantenere qualità e agilità in DevOps.

3.1.3 Reporting della Qualità per una Pipeline di CI/CD

In qualsiasi momento è importante conoscere il livello di qualità del software lungo tutto l'SDLC. Ciò fornisce feedback al team DevOps sulla qualità del codice e al product management sulla prontezza per il release. Integrare il reporting della qualità nella pipeline di CI/CD fornisce lo stato di sviluppo più recente. Ogni stadio della pipeline di CI/CD crea informazioni per il reporting della qualità. Alcuni dati di qualità

Quality in DevOps (CT-QDO) Syllabus

sono generati automaticamente dalla pipeline di CI/CD (per esempio l'esito positivo della build, lo stato superato/fallito dei test), mentre altri provengono dal testing manuale (per esempio user acceptance testing, testing esplorativo o copertura in uno strumento di test management). Queste informazioni creano una visione complessiva della qualità del software.

Il reporting della qualità dovrebbe avvenire in modo consistente per ogni ciclo della pipeline di CI/CD ed essere quindi automatizzato. I risultati dei test manuali dovrebbero essere inclusi nel reporting della qualità, tipicamente tramite integrazione con uno strumento di test management. Le informazioni sono raccolte in metriche e combinate in un quality report. Il quality report può essere consegnato con vari mezzi (per esempio una dashboard su una pagina wiki, uno strumento condiviso, un'email o un report basato su un template, oppure una combinazione di questi) (ISTQB® CTFL, v4.0).

Il reporting automatizzato della qualità richiede l'integrazione di diversi strumenti. Tra questi vi sono le funzionalità di logging, monitoraggio e reporting dei vari strumenti della pipeline di CI/CD (vedi *Sezione 4.1.2*). Gli strumenti di gestione delle attività e di test management registrano i risultati dei test manuali (per esempio il livello di qualità stimato delle funzionalità software o gli esiti superato/fallito dei test). Gli strumenti di monitoraggio in produzione forniscono la telemetria (vedi *Sezione 4.2.7*). I generatori di report dovrebbero integrarsi con tutti questi strumenti per creare report di facile comprensione per i vari stakeholder. Occorre creare una Single Source of Truth combinando le informazioni delle metriche provenienti da fonti diverse in un unico luogo accessibile (vedi *Sezione 3.1.1*).

Il processo di implementazione del reporting della qualità si articola in diversi passi:

1. determinare i key performance indicator (KPI) e le relative metriche da tracciare, quali qualità del codice, copertura, prestazioni, stabilità della build, tasso di successo della build, frequenza di deploy, tempo di recupero da un deploy fallito e vulnerabilità di sicurezza;
2. raccogliere i dati utilizzando le metriche scelte;
3. automatizzare la raccolta dei dati dai vari stadi della pipeline di CI/CD;
4. ripulire e normalizzare i dati per una migliore consistenza con strumenti di preparazione dei dati (vedi *Sezione 3.1.4*);
5. archiviare i dati raccolti in una base di dati centralizzata se gli strumenti non la forniscono;
6. visualizzare le metriche mediante dashboard, quali diagrammi, grafici e tabelle che mostrano le metriche chiave, con sezioni distinte per ciascuno stadio della pipeline di CI/CD;
7. abilitare il continuous monitoring nella dashboard per aggiornamenti in tempo reale;
8. assicurare l'accessibilità della dashboard agli stakeholder rilevanti, quali sviluppatori, tester, product owner, scrum master e manager.

Una dashboard di quality report può migliorare una pipeline di CI/CD. I benefici comprendono:

- tracciare con continuità le metriche chiave per una risposta e una risoluzione rapide;
- analizzare le tendenze delle prestazioni della pipeline di CI/CD per individuare i colli di bottiglia e ottimizzarne l'efficienza;
- verificare proattivamente lo stato di conformità alla sicurezza per assicurare che la codebase soddisfi gli standard di sicurezza;
- alimentare una cultura di trasparenza e collaborazione mediante dashboard condivise con gli stakeholder, allineando tutti sullo stato di qualità e sull'avanzamento del progetto;

- supportare le retrospettive con informazioni basate sui dati per individuare le aree di miglioramento.

3.1.4 Automazione della Gestione dei Dati di Test

Gestire manualmente i dati di test può ridurre il flusso e rallentare i cicli di feedback (vedi *Sezione 1.1.5*). I team DevOps utilizzano l'automazione della gestione dei dati di test per creare, aggiornare e gestire i dati di test all'interno della pipeline di CI/CD. L'uso di dati di test standardizzati aiuta ad accelerare i cicli di test, migliorare la copertura, ridurre gli errori, ridurre la non consistenza dei risultati dei test, migliorare la sicurezza dei dati e rispettare le normative sulla privacy.

Un'automazione efficace della gestione dei dati di test richiede la collaborazione tra tutti i ruoli del team DevOps per individuare su quali dati concentrarsi e quali pratiche utilizzare.

Gli aspetti chiave della gestione dei dati di test comprendono:

- Progettazione dei dati: analizzare e definire i dati di test necessari per un dato obiettivo del test (per esempio tipi, dimensioni e complessità dei dati).
- Generazione dei dati: creare o aggiornare i dati di test per consentire una copertura completa e una varietà di input (per esempio dati validi e non validi, casi limite e valori limite).
- Validazione dei dati: automatizzare la validazione per determinare l'accuratezza dei dati e la conformità a standard e normative.
- Organizzazione dei dati: organizzare e archiviare i dati di test in modo strutturato in un VCS e renderli disponibili su richiesta all'interno della pipeline di CI/CD o per il testing manuale.
- Anonimizzazione: mascherare o anonimizzare i dati di produzione prima di utilizzarli come dati di test, per tutelare privacy e confidenzialità.
- Aggiornamento e pulizia dei dati: aggiornare o ripristinare automaticamente i dati di test come parte della test automation.

I team DevOps applicano le seguenti pratiche nell'automazione della gestione dei dati di test:

- Creazione di dati di test sintetici: regole di formattazione dei dati, analisi della struttura dei dati di produzione o modelli GenAI pre-addestrati possono generare dati artificiali corrispondenti alle condizioni di test, senza utilizzare dati di produzione reali.
- Clonazione dei dati di test: creare dati di test validi e di grande volume per performance testing e load testing clonando dati esistenti.
- Subsetting dei dati di test: selezionare un sottoinsieme più piccolo e rappresentativo dei dati di produzione per il testing, mantenendo i riferimenti tra gli elementi di dati, così da ridurre le esigenze di archiviazione e accelerare il provisioning dei dati di test.
- Mascheramento dei dati di test: mascherare o modificare le Personally Identifiable Information (PII) provenienti dai dati di produzione per rendere i dati anonimi e tutelare l'identità della persona (per esempio nome, indirizzo, email, numero di telefono, indirizzo IP e dati di localizzazione).

Il team DevOps dovrebbe comprendere i rischi dell'automazione della gestione dei dati di test e gestirli mediante processi adeguati e strumenti ben scelti e configurati. Tali rischi comprendono:

- Copertura incompleta: i dati generati possono non riflettere la complessità del mondo reale.

- Problemi di utilizzo delle risorse: generare, archiviare e distribuire grandi quantità di dati di test può consumare risorse significative.
- Violazione delle regole di sicurezza e conformità: l'esposizione di informazioni sensibili dovuta a un mascheramento inadeguato dei dati può comportare sanzioni.

3.1.5 Analisi Statistica dei Risultati dei Test

L'esecuzione dei test produce vari tipi di risultati. L'uso a breve termine dei singoli risultati dei test consiste nell'informare gli stakeholder sulla qualità e sui rischi correlati, affinché possano maturare fiducia nel valore di business del software. L'uso a lungo termine dei risultati dei test consiste nell'ottenere un quadro più ampio della qualità del software, dei processi applicati e delle persone coinvolte. In entrambi i casi, applicando l'analisi statistica, il team DevOps può individuare tendenze e pattern relativi alla qualità del software e ai rischi residui, e utilizzare queste informazioni a supporto del processo decisionale. Per maturare fiducia, gli stakeholder hanno bisogno di metriche quali copertura dei requisiti, copertura del codice, densità dei difetti e mean time between failures.

Il team DevOps può utilizzare strumenti quali l'analisi predittiva basata su AI per prevedere i risultati futuri che il software produrrà. L'analisi predittiva supporta il miglioramento sia del processo sia delle persone coinvolte quando la situazione futura prevista non corrisponde a quella desiderata. L'analisi predittiva può evidenziare la necessità di un miglioramento di processo sulla base dell'efficacia dei test misurata, oppure può attivare l'allocazione di risorse maggiori o differenti.

Il testing A/B utilizza l'analisi statistica per confrontare più varianti di software mostrate a gruppi di utenti distinti, allo scopo di determinare quale variante sia la più preziosa. Un altro caso d'uso dell'analisi statistica è la valutazione dei risultati dei test non-funzionali, per esempio nel performance testing basato su profili di carico (Marselis et al., 2020).

L'analisi statistica è preziosa nell'analizzare i risultati di una singola test run o di più test run (per esempio per osservare le tendenze della qualità del software e determinare se la qualità aumenta dopo ogni iterazione). Il test process improvement può avvalersi dell'analisi statistica, che aiuta a individuare i test non consistenti e ad analizzare le keyword di test automation con tempi di esecuzione lunghi, supportando l'ottimizzazione del tempo di esecuzione dei test e dei costi correlati. Una dashboard può mostrare gli esiti dell'analisi statistica mediante appositi strumenti.

Gli strumenti per l'analisi statistica dovrebbero essere integrati nella pipeline di CI/CD.

3.1.6 Gestione degli Ambienti di Test Standardizzata, Controllata e Automatizzata

Una gestione degli ambienti di test standardizzata, controllata e automatizzata è una pietra angolare dello sviluppo software. Standardizzando gli ambienti di test le organizzazioni eliminano la variabilità causata da configurazioni non consistenti, assicurando che i test producano risultati affidabili e riproducibili. Questa consistenza riduce il tempo speso nell'indagare difetti derivanti da differenze ambientali, consentendo ai team DevOps di concentrarsi sui difetti reali del codice anziché sulle discrepanze di configurazione.

L'automazione snellisce il provisioning e la gestione degli ambienti di test, sostituendo processi manuali dispendiosi in termini di tempo con configurazioni rapide e ripetibili. Ciò consente cicli di test più rapidi, in cui gli ambienti possono essere creati e dismessi secondo necessità. Questo può comprendere la gestione dei dati di test (vedi *Sezione 3.1.4* e *Sezione 4.2.9*). I team DevOps possono testare e iterare sul

proprio codice più frequentemente, accelerando la consegna del software attraverso la pipeline di CI/CD e aumentando la produttività complessiva.

Gli ambienti controllati consentono un uso efficiente delle risorse assicurando che gli ambienti siano correttamente dimensionati e adattati alle esigenze degli specifici test. Ciò riduce il rischio di over-provisioning, in cui risorse eccessive vengono allocate senza necessità, e di under-provisioning, che può causare colli di bottiglia prestazionali. Il provisioning degli ambienti su richiesta abbassa i costi operativi e impiega le risorse solo quando servono.

L'uso dell'IaC per creare ambienti replicabili (vedi *Sezione 4.2.3*) elimina l'ambiguità sulle configurazioni, migliorando la collaborazione tra tester, sviluppatori e operations. L'uso dell'IaC supporta l'obiettivo di rimuovere il «wall of confusion». Questi ambienti fungono poi da modelli facilmente replicabili nel cloud, cosicché possono essere scalati agevolmente per far fronte a maggiori esigenze di testing o a team più numerosi.

3.2 Testing Automatizzato nei Team DevOps

Il regression testing è cruciale in DevOps per mantenere la qualità in presenza di deployment frequenti. Il regression testing comporta l'automazione dei test a copertura delle funzionalità nuove ed esistenti, bilanciando ottimizzazione e copertura in base ai livelli di rischio e ai livelli di qualità attesi.

Il testing delle API verifica le interfacce di sistema rispetto ad adeguatezza funzionale, affidabilità, efficienza delle prestazioni, sicurezza e altre caratteristiche di qualità. Offre feedback rapido e si integra bene nelle pipeline di CI/CD.

Nei modelli di sviluppo sequenziali la test automation avviene dopo lo sviluppo, mentre nello sviluppo software Agile avviene lungo tutto l'SDLC. DevOps si affida in misura rilevante alla test automation in tutte le fasi dell'SDLC, creando cicli di feedback rapidi e rispondendo a esigenze di test diverse con pratiche quali la containerizzazione.

Obiettivo Pratico QDO-3.2 (H2) Test Automation nei Team DevOps

Per questo obiettivo pratico l'esercizio di livello H2 supporta i discenti nell'apprendere come definire l'estensione e il ruolo della test automation in un contesto DevOps. L'attenzione non è posta sugli strumenti o sull'implementazione, bensì sulle decisioni strategiche riguardanti:

- che cosa dovrebbe essere automatizzato;
- dove l'automazione è essenziale, utile o facoltativa;
- che cosa non dovrebbe essere automatizzato e perché.

3.2.1 Regression Testing nella Pipeline di CI/CD

Lo scopo della pipeline di CI/CD è consegnare il software in produzione con una buona qualità. Il regression testing determina se i cambiamenti di codice non incidono negativamente sulle funzionalità esistenti dell'applicazione. Costituisce una rete di sicurezza critica, data la frequenza dei cambiamenti di codice e dei deployment. In pratica tutta la test automation nella pipeline di CI/CD è usata dapprima a copertura delle nuove funzionalità e può poi diventare regression testing.

I regression test utilizzano di norma più ambienti di test dello stesso tipo e vengono eseguiti in parallelo per ridurre i tempi di esecuzione. Ciascun ambiente di test fa sì che ogni tipo di test disponga di un numero adeguato di risorse e di integrazioni verso altri ambienti (vedi *Sezione 3.1.6*).

Tutti i test possono teoricamente essere eseguiti a ogni esecuzione della pipeline di CI/CD. Tuttavia, anche se ciò fosse sufficientemente rapido, sarebbe negativo per la sostenibilità economica e ambientale, poiché consuma risorse non necessarie (cioè tempo, costi ed energia). In pratica il team DevOps può creare più regression test suite con livelli di copertura differenti ed eseguirle per scopi diversi, con frequenze diverse o in base ai cambiamenti nel System Under Test. Alcuni regression test possono essere eseguiti prima dell'integrazione nella codebase, altri dopo l'integrazione e altri ancora durante la notte. Il team DevOps seleziona di norma i regression test in base al rischio (per esempio può far eseguire alla pipeline di CI/CD i test di componente sulla base di heat map che evidenziano le aree di codice modificate nell'ultimo commit). Tag e filtri consentono una scelta dinamica dei regression test. Un altro esempio è l'esecuzione dei regression test per le funzionalità esistenti interessate da un cambiamento.

La pipeline di CI/CD dovrebbe eseguire una regression test suite completa prima del rilascio del software. Il team DevOps dovrebbe bilanciare l'ottimizzazione del numero di regression test con la necessità di una buona copertura prima di un release. A seconda del livello di qualità atteso, i regression test possono essere eseguiti solo occasionalmente (per esempio per un test report di sicurezza). Il regression testing dovrebbe essere automatizzato il più possibile e far parte della progettazione della pipeline di CI/CD (vedi *Sezione 2.2.6*).

3.2.2 Punti Chiave del Testing delle API

Il testing delle API si concentra sulle interfacce tra sistemi e sul modo in cui essi comunicano. Comporta la verifica di adeguatezza funzionale, affidabilità, efficienza delle prestazioni, sicurezza e altre caratteristiche di qualità delle API. Le API sono la spina dorsale delle applicazioni moderne e testarle il prima possibile previene difetti costosi.

Il testing delle API è utilizzato per determinare se lo scambio di dati tra sistemi sia consistente, corretto, tempestivo e sicuro.

Aspetti chiave del testing delle API

- Testing funzionale: determina se l'API si comporta come atteso per tutti gli input e produce output corretti.
- Testing di affidabilità: determina se l'API funziona come atteso in scenari differenti.
- Performance testing: misura tempi di risposta, throughput e scalabilità.
- Testing di sicurezza: valida i meccanismi di autenticazione, autorizzazione e protezione dei dati.

Sono disponibili molti strumenti per il testing delle API e la maggior parte dei linguaggi di programmazione fornisce librerie utilizzabili allo scopo.

I benefici del testing delle API comprendono:

- feedback più rapido rispetto al testing della GUI;
- facile integrazione nelle pipeline di CI/CD;
- testing precoce nell'SDLC (cioè shift-left).

v1.0 Versione di Rilascio

Vi sono però anche delle sfide:

- assicurare una copertura accurata per API complesse;
- mantenere i test aggiornati rispetto all'evoluzione dei contratti delle API;
- gestire le dipendenze da API o servizi esterni.

Testing di contratti

Il testing di contratti è un tipo di testing di integrazione che testa ciascuna applicazione in isolamento per determinare se i messaggi che ognuna invia o riceve siano conformi a una comprensione condivisa documentata in un «contratto» (Fellows, 2022).

Il testing di contratti valida l'accordo (cioè il contratto) tra un provider e i suoi consumer. Il provider è accessibile tramite un'API e, invece di testare l'implementazione o la logica di business sottostante l'API, il testing di contratti verifica che l'API aderisca a una struttura, a un formato e a un insieme di aspettative definiti per i payload di richiesta e risposta, gli header, i codici di stato e i tipi di dato.

Il testing di contratti è importante nelle architetture a microservizi, dove numerosi servizi sviluppati in modo indipendente devono integrarsi in modo affidabile.

Il testing di contratti supporta il ciclo di feedback rapido in DevOps fornendo verifiche rapide e isolate per determinare se i cambiamenti non incideranno negativamente sui sistemi a valle.

Implementare il testing di contratti

1. Definire il contratto: iniziare specificando chiaramente la struttura e il comportamento attesi dell'API, compresi formati delle richieste, campi delle risposte, codici di stato e tipi di dato.
2. Scrivere i test di contratto: sulla base del contratto definito, i consumer specificano come intendono interagire con l'API. Tali specifiche guidano i test case che validano l'implementazione del provider. Sia il consumer sia il provider utilizzano lo stesso contratto per verificare che l'implementazione sia corretta dal lato del provider e che il consumer stia richiamando l'API correttamente.
3. Eseguirli nella pipeline di CI/CD: integrare i test di contratto nella pipeline di CI/CD per fornire feedback precoce e rilevare i cambiamenti che rompono la compatibilità prima del deployment.
4. Versionare e gestire i contratti: nelle organizzazioni con più team o servizi i contratti dovrebbero essere versionati e archiviati nel version control (vedi *Sezione 3.1.1*). Ciò consente di tracciare, esaminare e coordinare i cambiamenti tra i servizi.

Considerazioni e compromessi nel testing di contratti

- I test di contratto non coprono tutti gli aspetti funzionali del testing delle API e sono complementari ad altri tipi di test.
- I cambiamenti al contratto richiedono un attento coordinamento tra i diversi team.
- Il team di sviluppo deve investire in strumenti e definire standard condivisi.

3.2.3 Confrontare la Test Automation nei Diversi Modelli di SDLC

La test automation si applica a qualsiasi SDLC, sebbene i tempi e l'enfasi varino (*ISTQB® CTAL-TAE, v2.0*).

v1.0 Versione di Rilascio

I modelli di sviluppo sequenziali introducono la test automation oltre l'unit testing nella fase di testing, successiva al completamento dello sviluppo, e supportano un ciclo di feedback limitato tra le fasi di sviluppo e di testing. Lo sviluppo software Agile integra la test automation lungo tutto l'SDLC. In DevOps la test automation diventa un abilitatore centrale del CI/CD, con un ambito più ampio che copre vari livelli di test e tipi di test. Comporta un forte affidamento sull'IaC (vedi *Sezione 4.2.3*) per il provisioning degli ambienti di test e spesso include il testing in produzione mediante pratiche quali blue/green deployment e canary release (vedi *Sezione 2.2.4*).

Nei modelli di sviluppo sequenziali la test automation oltre gli unit test è generalmente limitata alle fasi finali dello sviluppo. Grazie all'approccio incrementale all'interno delle iterazioni dello sviluppo software Agile, vi è una maggiore attenzione a unit testing, testing di componente e testing di integrazione dei componenti. In DevOps l'affidamento sulla test automation è essenziale in tutti gli stadi della pipeline di CI/CD. Ciò crea cicli di feedback rapidi e automatizzati attraverso il testing pre-merge, il testing in produzione e il monitoraggio.

La test automation comporta sfide proprie a seconda dell'SDLC. La sfida principale della test automation in DevOps consiste spesso nel gestire la complessità dei diversi requisiti dei singoli livelli di test (per esempio componenti di sistema pienamente integrati nel testing end-to-end rispetto a servizi simulati per il testing di componente). La containerizzazione aiuta ad affrontare queste sfide fornendo ambienti isolati e riproducibili (vedi *Sezione 4.2.9*). Considerazioni culturali e di processo influenzano l'uso della test automation nei diversi SDLC. I modelli di sviluppo sequenziali vedono il testing come una fase distinta. Lo sviluppo software Agile promuove una mentalità collaborativa tra sviluppatori, tester e product owner riguardo al testing. DevOps rende ancora più sfumati i confini tradizionali dei ruoli, coinvolgendo i tester nelle review del codice, nel monitoraggio e nella configurazione dell'observability, e coinvolgendo gli sviluppatori nelle attività di testing e di monitoraggio.

3.3 Testing Manuale nei Team DevOps

Nonostante la forte enfasi sull'automazione in DevOps, il testing manuale resta essenziale per raccogliere informazioni critiche che il testing automatizzato non può fornire. Il testing manuale consente agli stakeholder di valutare direttamente l'usabilità, di maturare fiducia nella qualità del software e di valutare aspetti difficili da automatizzare, quali la user experience (UX). Il testing manuale svolge un ruolo cruciale all'interno delle pipeline di CI/CD e negli ambienti di produzione tramite il testing A/B e i feature toggle (vedi *Sezione 4.2.4*).

I principali approcci di testing manuale in DevOps comprendono testing esplorativo, crowd testing e quality hunting, ciascuno dei quali offre benefici peculiari per valutare dinamicamente la qualità del software. Questi approcci del test completano il testing automatizzato fornendo informazioni diversificate, assicurando che il software soddisfi sia le aspettative tecniche sia quelle degli utenti.

Obiettivo Pratico QDO-3.3 (H2) Implementare il Testing Manuale nei Team DevOps

Per questo obiettivo pratico l'esercizio di livello H2 supporta i discenti nell'apprendere come individuare, implementare e riportare in debriefing gli approcci di testing manuale in DevOps, per esempio con testing esplorativo, crowd testing o quality hunting, a complemento del testing automatizzato.

3.3.1 Esempi di Testing Manuale

Uno dei principi DevOps è automatizzare quanto più possibile. Ciò induce a credere che non vi sia spazio per il testing manuale nella pipeline di CI/CD ideale. Tuttavia il testing consiste nel raccogliere informazioni sulla qualità dell'oggetto di test, affinché gli stakeholder possano maturare fiducia nella soluzione, e tali stakeholder hanno quindi bisogno di informazioni di vario genere (Marselis et al., 2020).

Il testing automatizzato nella pipeline di CI/CD può fornire molte informazioni sul livello di qualità dell'oggetto di test (vedi *Sezione 2.2.4* e *Sezione 2.2.5*). Tuttavia il testing manuale resta un'attività essenziale per varie ragioni. La prima è che il testing manuale fornisce informazioni di tipo diverso rispetto al testing automatizzato (per esempio determinare l'usabilità del software è difficile da automatizzare ma agevole da svolgere con il testing manuale). La seconda è che alcuni stakeholder maturano una fiducia ulteriore nel livello di qualità del software se lo vedono con i propri occhi. La terza è che il testing manuale è flessibile, si avvale del giudizio umano e può risultare più efficiente in termini di costi rispetto al testing automatizzato, a seconda della specifica situazione.

Per queste ragioni il testing manuale è necessario, sia durante le attività della pipeline di CI/CD sia eventualmente durante l'esercizio (per esempio nel testing A/B oppure quando il software è stato rilasciato nell'ambiente di produzione ma non è ancora reso disponibile agli utenti, applicando i feature toggle) (vedi *Sezione 4.2.4*). Il testing manuale iniziale è il testing statico dei requisiti e/o delle user story (per esempio nelle sessioni di refinement con i 4-amigos, con rappresentanza di analisi di business, sviluppo, testing e operations) e le review del codice e dei prodotti di lavoro correlati nell'ambito delle pull request (PR) (vedi *Sezione 2.1.4*). Altri approcci rilevanti di testing manuale in DevOps sono il testing esplorativo (vedi *Sezione 3.3.2*), il crowd testing (vedi *Sezione 3.3.3*) e il quality hunting (vedi *Sezione 3.3.4*). Questi approcci del test forniscono un testing dinamico precoce che completa il testing automatizzato della pipeline di CI/CD.

3.3.2 Testing Esplorativo nella Pipeline di CI/CD

Alcuni ritengono che il testing esplorativo significhi semplicemente testare senza test case o procedure di test documentati in anticipo, ma vi è molto di più. Esplorare comporta un'indagine rigorosa e analitica (Hendrickson, 2013). L'esplorazione è la controparte dell'automazione: mentre il feedback automatizzato proviene dagli strumenti, il feedback dell'esplorazione proviene dalle persone (Clokier, 2017).

Nel testing esplorativo i test sono contemporaneamente progettati, eseguiti e valutati mentre il tester apprende sull'oggetto di test. Per la descrizione del testing esplorativo si veda (*ISTQB® CTFL*, v4.0). Il testing esplorativo può essere svolto come testing dinamico da un rappresentante degli utenti e da un membro del team per indagare il comportamento in situazioni specifiche e maturare fiducia. Sebbene i test esplorativi siano eseguiti manualmente, possono essere attivati dalla pipeline di CI/CD. Si possono utilizzare strumenti per supportare la creazione dei test case durante l'esplorazione (non prima), per supportare l'esecuzione dei test, per registrare i risultati dei test e per archiviare determinati tipi di evidenza (per esempio screenshot). I risultati dei test esplorativi dovrebbero essere registrati nella pipeline di CI/CD per essere inclusi nei quality gate e nel reporting del test, insieme ai risultati dei test automatizzati.

I test charter esplorativi possono essere definiti in una fase precoce (per esempio durante il refinement delle user story), così da avere un backlog di elementi di lavoro di testing come parte del backlog del team DevOps (vedi *Sezione 2.2.1*). I test charter possono anche essere creati quando emerge una determinata necessità di testing manuale (per esempio sulla base di risultati inattesi dei test automatizzati nella pipeline di CI/CD).

I test esplorativi possono essere eseguiti in vari ambienti di test, che vanno da uno specifico ambiente di test temporaneo creato dalla pipeline di CI/CD all'ambiente di staging o all'ambiente di produzione (per esempio per alcuni scopi specifici controllati dai feature toggle) (vedi *Sezione 2.2.5*).

L'IaC può essere utilizzata per esplorare l'impatto sul software di diverse versioni dell'infrastruttura. Questo approccio fornisce evidenze documentate di compatibilità tra le piattaforme, consentendo ai team di individuare i difetti specifici dell'infrastruttura e di risalire alle loro root cause prima del deployment.

3.3.3 Crowd Testing

Il crowd testing consiste nell'utilizzare un gruppo virtuale di persone, di norma esterne all'organizzazione e potenzialmente distribuite geograficamente, che rappresentano utenti reali, utilizzano dispositivi reali ed eseguono individualmente i test in contemporanea. Il crowd testing è utilizzato, per esempio, nel beta testing dopo una canary release, nel testing A/B, nella raccolta di feedback successiva al rilascio e nel testing su dispositivi reali. Il crowd testing applica di norma un approccio del test simile al testing esplorativo, in cui i membri della crowd ricevono un test charter che guida il loro testing senza predisporre in anticipo test case dettagliati.

La chiave del successo del crowd testing sta nel progettare ed eseguire numerosi test, raccogliendo rapidamente informazioni estese sulla qualità del software.

Il crowd testing privilegia l'ampiezza, per coprire molte persone differenti o molte combinazioni di software e hardware (come nel beta testing di videogiochi o di app mobile).

Un vantaggio principale del crowd testing è che si possono testare rapidamente molte situazioni e configurazioni diverse. Organizzazioni specializzate gestiscono la selezione dei crowd tester, distribuiscono i test charter e raccolgono i risultati dei test.

Uno svantaggio dell'impiego di tester esterni al team DevOps è che la preparazione richiede uno sforzo maggiore da parte del team DevOps (in particolare quando i test devono essere svolti in un determinato ordine) e può dover essere più approfondita (per esempio definendo persona specifiche) rispetto al caso in cui siano i membri del team DevOps stessi a svolgere i test.

Il reporting dei risultati del crowd testing può risultare impegnativo, per esempio a causa della grande quantità di dati raccolti in un intervallo di tempo breve. I team DevOps dovrebbero esaminare i defect report per prevenire feedback non giustificati. I crowd tester hanno bisogno di istruzioni chiare su quali informazioni raccogliere e su come registrare i risultati dei propri test. Alcune organizzazioni che organizzano il crowd testing supportano attivamente la definizione di tali istruzioni e lo svolgimento di queste attività di reporting.

I rischi e le sfide da considerare nell'applicare il crowd testing sono:

- **Confidenzialità:** quanto più la crowd è numerosa e quanto più si colloca al di fuori della sfera di influenza del team DevOps, tanto più diventa difficile gestire la confidenzialità.
- **Conoscenza:** non tutte le applicazioni si prestano al crowd testing dal punto di vista della conoscenza. Molte applicazioni utilizzate all'interno di un'organizzazione richiedono una conoscenza specifica dei suoi prodotti e processi per poter utilizzare il software.
- **Motivazione:** il metodo di remunerazione può influire sull'efficacia (per esempio i tester pagati «a difetto» cercheranno spesso difetti facili da trovare invece di cercare quelli più critici).

3.3.4 Eventi di Quality Hunting

Il software può essere consegnato con la giusta qualità al momento giusto solo se vi è una visione chiara di quale livello di qualità sia richiesto per ottenere il valore di business perseguito. Le persone coinvolte hanno bisogno di una visione chiara di quale sia attualmente il livello di qualità. Testing statico e testing dinamico insieme determinano il livello di qualità attuale. L'immagine della qualità dipende anche dall'impressione che gli stakeholder ricavano dalla propria esperienza personale. Per contribuire a costruire tale esperienza personale, un team DevOps può organizzare il quality hunting, un'attività collaborativa con un approccio di gamification (nota: è cosa diversa dal bug hunting) (Ruigrok et al., 2025).

Il quality hunting è un evento in cui squadre di quality hunting (per esempio della metà della dimensione di un team Agile) competono per fornire la valutazione più preziosa del livello di qualità del proprio oggetto di test in un tempo limitato. L'obiettivo principale del quality hunting è ottenere una visione condivisa del livello di qualità dell'oggetto di test. Facendo competere più squadre di quality hunting per ottenere la migliore visione del livello di qualità attuale, tali squadre sono stimolate a creare modi nuovi e innovativi di valutare la qualità e a fornire una buona comprensione di uno o più aspetti del livello di qualità. Durante il quality hunting le scoperte sono registrate in modo analogo alle informazioni di debriefing del testing esplorativo (vedi (*ISTQB® CTFL*, v4.0)). I difetti individuati durante il quality hunting sono gestiti secondo il processo di defect management dell'organizzazione. Le informazioni provenienti da più squadre di norma si completano a vicenda ed estendono le informazioni ottenute con il testing e con le altre attività di quality engineering (QE).

Gli stakeholder traggono beneficio dal quality hunting perché in breve tempo vengono indagate molte idee e nuove angolazioni sugli aspetti di qualità rilevanti, ottenendo così un quadro ampio e variegato della qualità del proprio sistema.

Il quality hunting mira a ottenere le informazioni più preziose sul livello di qualità del software. L'elemento di gioco consiste in un premio (di norma modesto) per la squadra di quality hunting che fornisce il report più informativo e/o l'angolazione più specifica sul livello di qualità. Il quality hunting è un'attività diversa e più ampia del bug hunting (introdotto in (*ISTQB® CT-MAT*, v2019)). Quest'ultimo è un approccio del test gamificato che motiva a trovare difetti, il che può indurre i tester a cercare difetti irrilevanti senza ottenere una visione del livello di qualità e del livello di rischio.

Il quality hunting è un'attività di testing manuale integrata nel passo della pipeline di CI/CD per testare il processo di business complessivo.

L'interazione tra le diverse squadre di quality hunting nel discutere i rispettivi risultati contribuisce alla cultura della collaborazione e dell'apprendimento in DevOps.

4 Strumenti e Pratiche in DevOps - 200 minuti

Parole chiave

fault injection, tolleranza ai guasti, hotfix, performance testing, testing di sicurezza, unit testing

Parole chiave specifiche di Quality in DevOps

artefatto di build, branch, strategia di branch, pipeline di CI/CD, chaos engineering, containerizzazione, analisi delle dipendenze, feature toggle, infrastructure as code, merge, observability, release management, strategia di rilascio, Software Bill of Materials, Source Code Management, version control, virtualizzazione

Obiettivi di Apprendimento del Capitolo 4:

4.1 Selezionare gli Strumenti a Supporto di DevOps all'Interno dell'Organizzazione

- QDO-4.1.1 (K1) Richiamare le capability degli strumenti in DevOps
- QDO-4.1.2 (K2) Spiegare gli strumenti a supporto della Quality Assurance e del testing in DevOps

4.2 Comprendere Tecnologie e Pratiche a Supporto della Qualità in DevOps

- QDO-4.2.1 (K1) Richiamare le attività non di testing all'interno della pipeline di CI/CD
- QDO-4.2.2 (K2) Spiegare le diverse strategie di rilascio principali in DevOps
- QDO-4.2.3 (K2) Riassumere i concetti di infrastructure as code
- QDO-4.2.4 (K1) Richiamare i feature toggle come modo per separare il deployment dal release
- QDO-4.2.5 (K1) Richiamare le strategie di branch
- QDO-4.2.6 (K1) Richiamare il chaos engineering
- QDO-4.2.7 (K1) Richiamare telemetria e observability
- QDO-4.2.8 (K1) Richiamare la Software Bill of Materials a supporto della gestione della configurazione
- QDO-4.2.9 (K1) Richiamare la containerizzazione

Obiettivi Pratici:

4.1 Strumenti a Supporto di DevOps all'Interno dell'Organizzazione

- QDO-4.1 (H2) Strumenti e Pratiche in DevOps

4.2.3 Infrastructure as code (IaC)

- QDO-4.2.3 (H0) Riassumere i Concetti di Infrastructure as Code (IaC)

4.1 Strumenti a Supporto di DevOps all'Interno dell'Organizzazione

Gli strumenti DevOps abilitano capability quali continuous discovery, continuous integration (CI), continuous testing, continuous delivery (CD), continuous deployment e continuous monitoring automatizzando i workflow, favorendo la collaborazione e migliorando l'affidabilità della consegna del software. Questi strumenti rientrano tipicamente in categorie quali version control, CI/CD, gestione della configurazione, management, monitoraggio e logging, testing, sicurezza e conformità, collaborazione e pianificazione. Tali strumenti snelliscono le attività essenziali. Un toolset DevOps ben integrato accresce velocità, efficienza e affidabilità, aiutando l'organizzazione a raggiungere i propri obiettivi di business mediante release software di alta qualità, rapide e sicure. Una fonte di informazioni completa e organica sugli strumenti DevOps è la DevSecOps tools periodic table (digital.ai, 2023).

Obiettivo Pratico QDO-4.1 (H2) Strumenti e Pratiche in DevOps

Per questo obiettivo pratico l'esercizio di livello H2 supporta i discenti nella scelta dei diversi strumenti per la pipeline di CI/CD, compresi sia gli strumenti di test automation sia gli altri strumenti di automazione.

4.1.1 Capability degli Strumenti

Questa sezione esamina come gli strumenti per il QE, che comprende QA e testing, siano integrati nei gruppi di attività del value stream, in particolare in continuous discovery, continuous integration (CI), continuous delivery (CD), continuous deployment e release on demand (*Sezione 2.2*). Di seguito una sintesi delle categorie principali, delle relative descrizioni e delle capability di questi strumenti.

1. Gestione delle informazioni: gestisce version control, collaborazione e tracciamento dei cambiamenti in documenti ed elementi di backlog, assicurando una Single Source of Truth (SSOT) (per esempio documentazione di progetto e di prodotto, requisiti, defect report e attività)
Capability: version control, tracciabilità, trasparenza e controllo dei cambiamenti
2. Source Code Management: gestisce version control, collaborazione e tracciamento dei cambiamenti nelle codebase, assicurando una SSOT
Capability: version control, branch, merge e risoluzione dei conflitti
3. Continuous integration: automatizza l'integrazione dei cambiamenti di codice in repository condivisi di software funzionante, favorendo il rilevamento precoce dei difetti e cicli di feedback continui
Capability: server di CI, pipeline di build automatizzate e testing di integrazione
4. Continuous deployment: automatizza il rilascio del codice validato su qualsiasi ambiente, riducendo gli errori manuali e assicurando deployment consistenti
Capability: pipeline di deployment, automazione del deployment, meccanismi di roll forward e di rollback
5. Gestione degli ambienti e dell'infrastruttura: automatizza il provisioning e la configurazione di ambienti consistenti tra sviluppo, testing e produzione
Capability: infrastructure as code (IaC), provisioning degli ambienti e gestione della configurazione

Quality in DevOps (CT-QDO) Syllabus

6. Gestione degli artefatti di build: agevola l'archiviazione, il recupero e la distribuzione efficienti di artefatti di build, dipendenze e pacchetti, assicurando consistenza e ripetibilità tra gli ambienti
Capability: archiviazione degli artefatti di build, gestione delle dipendenze, analisi delle dipendenze, scanning delle vulnerabilità e distribuzione sicura
7. Qualità e sicurezza del codice: migliora la qualità e la resilienza del software rilevando precocemente difetti e vulnerabilità e assicurando l'aderenza alle pratiche di codifica sicura lungo tutto lo sviluppo
Capability: analisi statica, Static e Dynamic Application Security Testing (SAST/DAST), analisi della composizione del software e delle dipendenze (SCA), validazione della sicurezza della supply chain ((ISTQB® CT-SEC, v1.0)), unit testing e valutazione della copertura del codice
8. Integrazione e test automation: valida le interazioni tra i componenti del sistema e le funzionalità complessive, automatizzando al contempo i test per ragioni di efficienza
Capability: testing di contratti, testing delle API, testing di integrazione dei componenti, testing della UI e framework di test automation riutilizzabili
9. Performance testing e testing di sicurezza: valuta scalabilità, tolleranza ai guasti e sicurezza dell'applicazione simulando condizioni di carico ed effettuando lo scanning delle vulnerabilità (ISTQB® CT-PT, v2018) e (ISTQB® CT-SEC, v1.0)
Capability: performance testing, load testing, stress testing, spike testing, testing di volume, testing di endurance, scanning delle vulnerabilità e penetration testing
10. Telemetria (monitoraggio e observability): fornisce informazioni in tempo reale sullo stato di salute dell'applicazione e dell'infrastruttura, aiutando a individuare proattivamente i difetti
Capability: monitoraggio delle prestazioni, aggregazione dei log, analisi dei log, tracing distribuito e alerting
11. Chaos engineering e fault injection: simula failure per testare la resilienza e la tolleranza ai guasti del sistema, individuando le debolezze sotto stress o in qualsiasi condizione inattesa
Capability: fault injection, testing di resilienza e validazione della tolleranza ai guasti
12. Dati di test: generano, gestiscono e mettono in sicurezza i dati di test a supporto di processi di test affidabili e conformi
Capability: generazione di dati di test sintetici, anonimizzazione, mascheramento, subsetting e clonazione di dati di test simili a quelli di produzione, provisioning su richiesta dei dati di test e conformità alle normative sulla privacy (per esempio il General Data Protection Regulation (GDPR))
13. Test management: organizzano e controllano il processo di test gestendo test case, test plan, risultati dei test e tracciabilità
Capability: redazione e version control dei test case, creazione e schedulazione dei test plan, tracciabilità tra requisiti, test case e difetti, reporting e dashboarding in tempo reale, integrazione con pipeline di CI/CD e sistemi di tracciamento dei difetti (vedi (ISTQB® CTFL, v4.0))
14. Collaborazione e comunicazione: accrescono la produttività e l'allineamento del team mediante strumenti di comunicazione e di gestione dei progetti snelli
Capability: messaggistica in tempo reale, gestione dei progetti, condivisione dei documenti e versionamento

15. Qualità e affidabilità dell'esercizio: supportano gli ambienti di produzione assicurando l'affidabilità operativa mediante backup, ripristino e gestione degli incidenti (AXELOS, 2019)

Capability: backup dei dati, ripristino, gestione degli incidenti e scanning delle vulnerabilità

Queste categorie formano un framework strutturato all'interno della pipeline di CI/CD, in cui ciascuno strumento assolve una finalità peculiare per creare un ambiente di sviluppo e di esercizio continuo, affidabile ed efficiente. Allineata al workflow DevOps, ogni categoria si integra in modo fluido nel loop DevOps, alimentando cicli di feedback per il miglioramento continuo dallo sviluppo al deployment e alla produzione.

4.1.2 Strumenti a Supporto della Quality Assurance (QA) e del Testing

In un ambiente DevOps gli strumenti a supporto di QA e testing sono fondamentali per consegnare rapidamente software affidabile, performante e sicuro. Questi strumenti si integrano in modo fluido nella pipeline di CI/CD, incorporando la qualità in ogni fase dell'SDLC (vedi *Sezione 2.2*). Automatizzare e snellire le varie attività di test mette i team DevOps nelle condizioni di rilevare precocemente i difetti, mantenere la qualità del codice e fornire informazioni sulla qualità del software. Gli strumenti possono essere raggruppati per funzione.

Durante le fasi di codifica e integrazione gli strumenti si concentrano sull'assicurare il controllo e l'integrità della codebase e dei suoi componenti (vedi *Sezione 4.2.8*).

- Gli strumenti di analisi statica e di qualità del codice valutano il codice sorgente senza eseguirlo, contribuendo a far rispettare gli standard di codifica, a rilevare potenziali difetti e a individuare code smell o vulnerabilità di sicurezza.
- Gli strumenti di unit testing verificano le singole unità di codice, fornendo feedback rapido agli sviluppatori.
- Gli strumenti di analisi delle dipendenze esaminano le librerie di terze parti alla ricerca di rischi di sicurezza, pacchetti obsoleti o problemi di licenza, contribuendo a mantenere una codebase sicura e conforme.
- Gli strumenti di testing di integrazione e di testing delle API verificano che i componenti o i servizi collegati funzionino correttamente insieme e soddisfino le aspettative sulle interfacce.

Man mano che il lavoro sul software si avvicina al deployment, assumono rilievo gli strumenti progettati per il testing funzionale e il testing della UI, il performance testing, il testing di sicurezza e la gestione dei dati di test.

- Gli strumenti di testing funzionale e di testing della UI simulano il comportamento dell'utente per validare che il software soddisfi i requisiti funzionali dal punto di vista dell'utente finale.
- Gli strumenti di performance testing valutano il comportamento di un sistema in diverse condizioni di carico, individuano i colli di bottiglia prestazionali e assicurano la scalabilità.
- Gli strumenti di testing di sicurezza rilevano proattivamente le vulnerabilità, quali difetti di injection o configurazioni errate, riducendo l'esposizione alle minacce.
- Gli strumenti di gestione dei dati di test generano e gestiscono dati di test realistici, anonimizzati o sintetici, assicurando che le condizioni di test riflettano le condizioni reali senza violare la data privacy.

Quality in DevOps (CT-QDO) Syllabus

Framework e strumenti a supporto di automazione, test management e provisioning degli ambienti svolgono un ruolo essenziale nel rendere efficace il testing. (Vedi (*ISTQB® CTFL*, v4.0), sezione 6.1 Tool Support for Testing)

- Gli strumenti di provisioning degli ambienti automatizzano la creazione di ambienti di test che rispecchiano le configurazioni di produzione, supportando un'esecuzione dei test affidabile e consistente.
- Gli strumenti di gestione della configurazione assicurano la consistenza delle impostazioni degli ambienti e delle configurazioni applicative tra gli ambienti di sviluppo, testing e produzione. (Vedi (*ISTQB® CTFL*, v4.0), sezione 5.4 Configuration Management) Questa consistenza riduce gli errori e accresce l'accuratezza dei risultati dei test, supportando processi di test affidabili e ripetibili.

Nel loro insieme questi strumenti supportano i team DevOps nel raggiungere e mantenere la qualità richiesta.

4.2 Tecnologie e Pratiche a Supporto della Qualità in DevOps

Le tecnologie e le pratiche in DevOps supportano la qualità integrando attività non di testing, strategie di branch efficaci, automazione dell'infrastruttura, release management e feature toggling. Le attività non di testing, quali build, packaging e deployment, automatizzano passi chiave all'interno della pipeline di CI/CD, mentre le strategie di rilascio come canary release e blue-green deployment controllano il processo di rilascio. L'IaC consente la consistenza degli ambienti, i feature toggle disaccoppiano il release dal deployment, il release management e le strategie di branch ottimizzano la collaborazione. Queste tecnologie e pratiche creano un ambiente coeso che assicura consistenza, affidabilità, sicurezza e rapidità nella consegna del software.

4.2.1 Attività non di Testing all'Interno di una Pipeline di CI/CD

Una pipeline di CI/CD automatizza diversi processi dell'SDLC. Suddividere questi processi in più stadi sequenziali assicura una consegna fluida ed efficiente. Ogni stadio prevede attività specifiche che comprendono attività non di testing critiche per il successo della pipeline di CI/CD.

Segue un elenco di tali attività non di testing: (vedi *Sezione 2.1.4* e *Sezione 3.1.1*)

1. Source Code Management: sovrintendere ai commit di codice e al version control all'interno di un repository condiviso per mantenere collaborazione e organizzazione.
2. Branch e merge: svolgere attività quali branch, merge e creazione di pull request (PR) per organizzare ed esaminare efficientemente i cambiamenti di codice.
3. Gestione degli artefatti di build e delle dipendenze: (vedi *Sezione 4.2.8*) risolvere le dipendenze software, compilare il codice e generare artefatti di build versionati. Tali artefatti di build sono archiviati per gli stadi successivi della pipeline di CI/CD.
4. Risoluzione dei conflitti e standard di codifica: (vedi *Sezione 3.1.6* e *Sezione 4.2.3*) affrontare i conflitti di codice, effettuare il provisioning degli ambienti di integrazione ed eseguire l'analisi statica per far rispettare gli standard di codifica e assicurare la qualità del codice.
5. Configurazione della pipeline di CI/CD: predisporre le pipeline di CI/CD e automatizzare il provisioning di ambienti isolati e consistenti, spesso ottenuto mediante strumenti di

Quality in DevOps (CT-QDO) Syllabus

containerizzazione. Inoltre gestire le configurazioni specifiche dell'ambiente e i feature toggle tramite argomenti da riga di comando o variabili d'ambiente, per supportare flessibilità e scalabilità (vedi *Sezione 4.2.4*).

6. Release management: (vedi *Sezione 2.2.5*) coordinare e approvare i release, gestire le strategie di rollback (vedi *Sezione 4.2.2*) e assegnare tag o etichette alle versioni. Ciò comprende l'implementazione di strategie di deployment quali il blue-green deployment o le canary release.
7. Monitoraggio e feedback: utilizzare strumenti per monitorare lo stato di salute del sistema e istituire cicli di feedback per il miglioramento continuo. Ciò comporta la configurazione dei sistemi di logging e di telemetria, l'analisi delle metriche per valutare prestazioni e tendenze dell'applicazione e la predisposizione di alert per gli eventi critici. (vedi *Sezione 4.2.7*) Queste attività forniscono informazioni essenziali sullo stato di salute e sulla stabilità dell'applicazione dopo il deployment in produzione.

4.2.2 Principali Strategie di Rilascio

Le strategie di rilascio DevOps delineano una consegna controllata, efficiente e affidabile degli aggiornamenti software e delle nuove funzionalità agli utenti. Tali strategie di rilascio si concentrano sui punti seguenti:

- minimizzare i rischi riducendo i tempi di fermo, consentendo un'esposizione graduale ai cambiamenti e rilevando proattivamente gli incidenti (AXELOS, 2019);
- ottimizzare i processi di deployment, aumentando la velocità di release, eliminando le attività ripetitive e accorciando il tempo dallo sviluppo alla produzione;
- consegnare valore al cliente senza soluzione di continuità, assicurando una UX ininterrotta e consentendo ai team operativi una rapida risoluzione degli incidenti (AXELOS, 2019).

Comprendere queste strategie di rilascio può aiutare le organizzazioni a selezionare quella più adatta in base alle proprie esigenze specifiche e/o operative.

DevOps prevede diverse strategie di rilascio principali, ciascuna con finalità e casi d'uso propri. Queste strategie comprendono:

1. canary release (vedi *Sezione 2.2.4*);
2. blue-green deployment (vedi *Sezione 2.2.4*);
3. Rolling release: sostituire gradualmente le istanze del sistema con le versioni aggiornate, assicurando la continuità della disponibilità del servizio. Utilizzato di frequente nei sistemi su larga scala, questo approccio mitiga i rischi di disservizi estesi.
4. Phased rollout: rilasciare gli aggiornamenti in modo incrementale per regione, gruppo di utenti o altro criterio di segmentazione. Ciò consente monitoraggio e risoluzione dei problemi per fasi successive.
5. Progressive delivery: combinare elementi delle canary release e dei phased rollout, rilasciando gli aggiornamenti in modo incrementale a utenti o regioni mirati, per una validazione precisa e per la raccolta di feedback.
6. Hotfix deployment: un release urgente per risolvere difetti critici in produzione, che privilegia la rapidità rispetto a un testing accurato.

7. Dark launching (vedi *Sezione 2.2.4*): rilasciare le funzionalità in produzione mantenendole però inaccessibili agli utenti. Ciò consente ai team di testare e monitorare le prestazioni in condizioni reali senza incidere sulla UX.

Le principali strategie di rilascio variano in base alle esigenze e al contesto specifici di ciascuna organizzazione, dato che ne esistono molte. Tuttavia le strategie sopra indicate rappresentano le scelte più essenziali e allineate alle moderne pratiche DevOps. Inoltre le organizzazioni combinano spesso più strategie di rilascio per ottimizzare le proprie attività quotidiane (per esempio rilasciando un hotfix mediante un phased rollout).

4.2.3 Infrastructure as code (IaC)

L'IaC gestisce ed effettua il provisioning dell'infrastruttura mediante file di definizione leggibili dalla macchina, trattando l'infrastruttura come software. Utilizza modelli descrittivi e codice sotto version control per garantire consistenza e ripetibilità. La QA nell'IaC comporta testing statico automatizzato, review tra pari e validazione della configurazione per determinare il grado di affidabilità, sicurezza e conformità prima del deployment.

Con l'IaC i team modificano il codice sorgente IaC correlato invece di apportare cambiamenti direttamente all'ambiente di destinazione. Questa pratica si integra in modo fluido nei workflow DevOps, consentendo provisioning e gestione automatizzati e assicurando consistenza, scalabilità ed efficienza. L'IaC supporta la CD rendendo possibili ambienti infrastrutturali riproducibili e prevedibili (vedi *Sezione 3.1.6*).

L'IaC si fonda su diversi concetti chiave per produrre efficacemente i propri benefici. Anzitutto l'IaC utilizza approcci dichiarativi e imperativi per definire l'infrastruttura. L'approccio dichiarativo si concentra sulla specifica dello stato desiderato dell'infrastruttura (per esempio assicurare che tre server siano sempre in esecuzione), mentre l'approccio imperativo fornisce istruzioni dettagliate su come raggiungere tale stato (per esempio installare il software e configurare le impostazioni).

Un altro concetto critico è l'idempotenza (Kundu, 2019), che assicura che gli script IaC producano gli stessi risultati anche se applicati più volte, prevenendo la deriva della configurazione (Moustakis, 2024). L'IaC si integra in modo fluido con i sistemi di gestione della configurazione, consentendo ai team di tracciare i cambiamenti, collaborare efficientemente e tornare a configurazioni precedenti se necessario (vedi *Sezione 3.1.1* e *Sezione 4.1.1*). L'IaC elimina la necessità di attività manuali ripetitive mediante l'automazione, assicurando consistenza e riducendo l'errore umano. Supporta la parità tra ambienti, mantenendo configurazioni identiche tra gli ambienti di sviluppo, testing e produzione. Ciò riduce i casi del tipo «sulla mia macchina funziona» e consente transizioni più agevoli tra i livelli di test.

Obiettivo Pratico QDO-4.2.3 (H0) Riassumere i Concetti di Infrastructure as Code (IaC)

Per questo obiettivo pratico l'esercizio di livello H0 supporta i discenti nella comprensione del concetto di IaC.

4.2.4 Feature Toggle

L'uso dei feature toggle può aiutare i team DevOps a modificare il comportamento del sistema per consegnare rapidamente ma in sicurezza nuove funzionalità agli utenti senza modificare il codice (Fowler, 2017). Fornisce un approccio strutturato alla gestione del rilascio delle funzionalità, consentendo

release incrementali, mitigazione del rischio e sperimentazione controllata. In un ambiente di produzione i feature toggle definiscono quali utenti o ambienti possono accedere a una determinata funzionalità. Per esempio una funzionalità può essere disabilitata per la maggior parte degli utenti e attivata per i tester, assicurando una validazione controllata prima di un deployment più esteso. L'uso dei feature toggle supporta la separazione del deployment dal release.

I feature toggle sono classificati in base all'uso previsto:

- Release toggle: abilitano nuove funzionalità in linea con gli obiettivi di release on demand (per esempio una canary release), vedi *Sezione 2.2.4*
- Operational toggle: consentono il controllo in tempo reale del comportamento del sistema
- Experiment toggle: supportano il testing A/B esponendo funzionalità differenti a segmenti di utenti
- Permission toggle: controllano l'accesso alle funzionalità per utenti o ruoli specifici

I feature toggle introducono complessità. La loro gestione richiede tracciamento e organizzazione sistematici per evitare che se ne accumulino in numero eccessivo nel tempo. Poiché i feature toggle aggiungono logica condizionale alla codebase, è essenziale monitorare, documentare e dismettere quelli obsoleti per mantenere la chiarezza del sistema. I team DevOps dovrebbero testare le implementazioni dei feature toggle sia in posizione attiva sia disattiva e testare le funzionalità interdipendenti per ridurre il rischio.

4.2.5 Strategie di Branch

Il branch consente lo sviluppo parallelo all'interno dei team DevOps e tra di essi, fornendo più flussi di lavoro contemporanei e minimizzando conflitti e difetti di integrazione. L'integrazione di tali cambiamenti di codice da un ramo a un altro è detta merge. Un conflitto di merge si verifica quando entrambi i rami modificano la stessa porzione di codice. La probabilità di conflitti di merge dipende da molteplici fattori (per esempio l'entità dei cambiamenti da integrare o il numero di sviluppatori che introducono cambiamenti). Risolvere tali conflitti può richiedere molto tempo e molto impegno.

Sebbene i VCS supportino branch, merge e rilevamento dei conflitti, i team DevOps dovrebbero implementare strategie di branch adeguate per ridurre il rischio di conflitti assicurando al contempo collaborazione e qualità.

Poiché tutto il codice e il testware che contribuiscono allo sviluppo software risiedono in un VCS (vedi *Sezione 3.1.1*), le strategie di branch si applicano anche a essi e devono essere comprese da ogni membro del team DevOps.

Tra le molteplici strategie di branch definite, i team DevOps applicano di norma le due seguenti per conseguire la capability di CD:

- Trunk-based development: dopo il superamento dei test durante il commit stage, ogni cambiamento viene inviato direttamente al ramo principale (cioè quello in passato denominato trunk o master). Questo approccio richiede un team maturo e un elevato livello di test automation all'interno della pipeline di CI/CD.
- Feature branch: sono creati per cambiamenti specifici ed eliminati dopo l'integrazione. Una PR è utilizzata per integrare i cambiamenti nel ramo principale oppure per ottenere feedback dagli altri membri del team DevOps durante lo sviluppo (vedi *Sezione 2.1.4*). Questi feature branch

dovrebbero avere una vita breve, al massimo un paio di giorni, con l'obiettivo di ridurla a poche ore.

4.2.6 Chaos Engineering

Il chaos engineering è una disciplina che consiste nello sperimentare su un sistema per maturare fiducia nella sua capacità di sopportare condizioni turbolente in produzione (Ranganathan, 2023). Questa pratica è cruciale per determinare la resilienza e l'affidabilità dei sistemi software iniettando intenzionalmente failure e osservando come il sistema reagisce.

Il chaos engineering simula failure a vari livelli, da singole istanze di elaborazione (per esempio macchine virtuali o container che eseguono applicazioni) fino a intere regioni geografiche dell'infrastruttura cloud, dove più data center operano insieme (Netflix, 2025). Questi esperimenti aiutano i team a comprendere come i servizi si comportano durante i disservizi e a determinare la continuità del servizio. Le informazioni ricavate da questi esperimenti portano a miglioramenti nella progettazione del sistema, nelle pratiche operative e nelle strategie di risposta agli incidenti.

Il chaos engineering sottolinea l'importanza di comprendere il comportamento del sistema sotto stress. I team DevOps possono scoprire difetti nascosti che potrebbero non emergere durante il normale esercizio o in un ambiente di test. Questa disciplina proattiva individua le potenziali debolezze prima che incidano sugli utenti, accrescendo la qualità del software.

Il chaos engineering è reso possibile da una pipeline di CI/CD altamente efficace, rapida e affidabile, che consente di rilasciare i cambiamenti in produzione e di effettuare il rollback senza difficoltà, minimizzando l'impatto sugli utenti. I provider di cloud pubblico offrono di norma strumenti di fault injection per supportare il chaos engineering in modo controllato.

4.2.7 Telemetria e Observability

Quando il software è rilasciato nell'ambiente di produzione, il continuous monitoring raccoglie informazioni sulla qualità del software e, quando la qualità non progredisce come atteso, vengono avviate azioni di controllo. Il continuous monitoring osserva il software nel suo ambiente, comprendendo altro software, hardware e le persone coinvolte. In molte organizzazioni il continuous monitoring parte dall'ambiente di produzione, ma può essere rilevante per qualsiasi altro ambiente, in particolare per gli ambienti di test. Telemetria (raccolta dei dati) e observability (analisi dei dati) abilitano il continuous monitoring (che effettua il confronto con i KPI).

L'observability utilizza logging, tracing e metriche per fornire informazioni e creare una visione coerente dello stato del software. Queste informazioni sono utilizzate a fini di manutenzione reattiva e proattiva, auditability, controllabilità e debugging (Marselis et al., 2020). Le informazioni sono raccolte automaticamente tramite la telemetria. La telemetria raccoglie misurazioni per le caratteristiche di qualità funzionali e non-funzionali, che rispondono a finalità e obiettivi differenti.

Esempi di telemetria sono:

- la telemetria di produzione misura come il software funziona, in particolare quando si utilizza infrastruttura cloud (per esempio per misurare utilizzo della memoria, carico della CPU, latenza e conteggio delle richieste);
- la telemetria sugli utenti misura l'interazione degli utenti, fornendo informazioni essenziali sul successo del software (per esempio tracciamento dei click per l'uso delle funzionalità,

tracciamento del tasso di completamento dei moduli, analisi dei flussi di navigazione e monitoraggio della durata delle sessioni);

- la telemetria sugli endpoint e altri approcci di monitoraggio della sicurezza abilitano il rilevamento delle minacce, l'analisi forense e il reporting di conformità.

Il monitoraggio (cioè l'uso di telemetria e observability) può applicare l'Artificial Intelligence (AI), per esempio il machine learning, ai dati di log, per comprendere meglio e più agevolmente il comportamento del software e rilevare eventuali anomalie.

Si noti che la telemetria sul team può essere utilizzata per misurare la maturità del team DevOps (per le metriche DORA vedi la *Sezione 1.1.3*) e per suggerire come il team DevOps possa migliorare (per maggiori informazioni sull'apprendimento del team vedi la parte finale della *Sezione 2.2.5*).

4.2.8 Software Bill of Materials (SBOM)

Nello sviluppo del software i team DevOps utilizzano spesso componenti di terze parti. Tra questi vi sono framework, middleware e librerie che forniscono funzionalità specifiche, oltre a codice proveniente da altri team DevOps. Il riutilizzo accelera lo sviluppo e porta a uno sviluppo software consistente all'interno delle organizzazioni. La riutilizzabilità (come parte della manutenibilità) (vedi (*Systems and software engineering* (25010), 2023)) è un'importante caratteristica di qualità e un segno di componenti costruiti correttamente. Quando si riutilizzano componenti, è essenziale per la gestione dell'SDLC, per la manutenzione futura e per la trasparenza disporre di informazioni su ciascun componente — compresi i componenti che il team DevOps crea autonomamente — e sulle relazioni tra i componenti.

Sono necessarie le seguenti informazioni:

- identificazione del componente (per esempio nome o numero);
- versione corrente del componente;
- autore e/o titolare del componente;
- funzionalità del componente;
- consistenza del componente con gli altri componenti;
- dipendenze del componente da altri componenti;
- conflitti della versione corrente con le versioni precedenti;
- hash crittografico o hash crittografici per autenticare il componente o i componenti;
- condizioni di licenza del componente.

Queste informazioni sono registrate in un SBOM (Marselis et al., 2020). I team DevOps dovrebbero mantenere un SBOM per il proprio software utilizzando strumenti integrati nella pipeline di CI/CD. Per assicurare che l'SBOM sia aggiornato, esso dovrebbe far parte della definition of done del team DevOps. Oltre che alla manutenibilità, un SBOM contribuisce alla gestione delle vulnerabilità — che fa parte della sicurezza — associando le vulnerabilità note ai componenti utilizzati. Ogni volta che viene rilevata una vulnerabilità la pipeline di CI/CD si interrompe, per impedire il deployment di software non sicuro. Successivamente il team DevOps può intraprendere le azioni necessarie a risolvere tali vulnerabilità.

4.2.9 Containerizzazione

La containerizzazione è una forma leggera di virtualizzazione che impacchetta un'applicazione e le sue dipendenze in un'unica unità eseguibile denominata container.

I container consentono al team DevOps di costruire ambienti consistenti per ogni livello di test ed eliminano le insidie del testing in locale, come il «sulla mia macchina funziona» (vedi *Sezione 3.1.6*). Un altro beneficio è che consentono di eseguire più test in parallelo senza interferenze reciproche, mantenendo puliti i dati di test (vedi *Sezione 3.1.4*). Un container può essere rilasciato su richiesta e rimosso una volta terminata la test run. Ciò aiuta a effettuare il provisioning degli ambienti quando sono effettivamente in uso e a fornire rapidamente ambienti aggiuntivi quando la domanda cresce. Poiché i container sono definiti e archiviati come codice in una codebase condivisa, qualsiasi cambiamento agli ambienti apportato centralmente può essere applicato automaticamente agli ambienti futuri con la stessa configurazione.

La containerizzazione comporta alcune sfide note. Poiché ogni container è una rappresentazione pulita dell'ambiente, gestire il testing di applicazioni con stato può risultare difficile e richiede passi di configurazione aggiuntivi. La sicurezza può costituire un elemento di attenzione per via dell'uso di componenti di terze parti e richiede una gestione diligente delle dipendenze (vedi *Sezione 4.2.8*).

5 Termini specifici di Quality in DevOps

Termine	Definizione
analisi delle dipendenze	Processo di individuazione e analisi delle dipendenze software volto a ridurre i rischi introdotti da componenti esterni.
analisi statistica	Tecnica per raccogliere, analizzare, interpretare, presentare e trarre conclusioni dai dati.
artefatto di build	Prodotto di lavoro generato dal processo di build e utilizzabile per il deployment.
blue-green deployment	Strategia di deployment in cui si mantengono due ambienti di produzione identici, uno attivo e uno inattivo, per testare una nuova versione del software nell'ambiente inattivo prima di dirottare tutto il traffico dall'ambiente attivo.
branch	Processo consistente nel lavorare in modo indipendente sui cambiamenti software duplicando il codice sorgente sotto version control in un ramo relativo al cambiamento e integrandolo successivamente in un altro ramo.
CALMS	Abbreviazione che sta per culture, automation, lean, measurement e sharing, usata per strutturare le pratiche DevOps.
change fail percentage	Numero di cambiamenti rilasciati che determinano failure, diviso per il numero totale di cambiamenti rilasciati, espresso in percentuale.
change lead time	Tempo che intercorre dal momento in cui viene effettuato un commit fino a quando esso raggiunge la produzione.
chaos engineering	Pratica consistente nell'iniettare casualmente failure per raccogliere informazioni sulla resilienza del sistema.
ciclo di feedback	Scambio continuo di informazioni volto a migliorare prodotti, servizi e processi.
code smell	Caratteristica del codice sorgente che indica un potenziale problema di progettazione o di manutenibilità, senza necessariamente impedire al codice di funzionare.
containerizzazione	Forma leggera di virtualizzazione che impacchetta un'applicazione e le sue dipendenze in un'unica unità eseguibile denominata container.
continuous delivery (CD)	Procedura automatizzata di sviluppo software in cui i cambiamenti di codice sono automaticamente costruiti, testati e resi rilasciabili in produzione.
continuous deployment	Procedura automatizzata di rilascio del software in cui i cambiamenti di codice che hanno superato tutti i test sono rilasciati automaticamente in produzione senza intervento manuale.
continuous discovery	Processo continuativo di user research e di raccolta del feedback degli stakeholder, volto a prendere decisioni informate, migliorare il prodotto e assicurare la consegna di valore reale agli utenti.
continuous learning and experimentation	Principio DevOps che alimenta una cultura ad alta fiducia orientata al miglioramento continuo, incoraggiando i team a sperimentare, ad assumere rischi calcolati e ad apprendere dai successi e dai failure.
continuous monitoring	Processo automatizzato di raccolta degli indicatori di prestazione per comprendere in tempo reale il comportamento degli utenti e del sistema.

Quality in DevOps (CT-QDO) Syllabus

Termine	Definizione
dark launch	Strategia di deployment in cui una nuova funzionalità o un cambiamento software è rilasciato senza essere esposto agli utenti, consentendo ai team di testare e monitorare le prestazioni in produzione.
DevOps	Mentalità, cultura e insieme di pratiche tecniche che supportano l'integrazione, l'automazione e la collaborazione necessarie a sviluppare ed esercitare efficacemente un sistema.
feature toggle	Meccanismo per modificare il comportamento del sistema senza modificare il codice.
file binario	File risultante dalla compilazione di codice sorgente scritto in un linguaggio compilato.
flusso (flow)	Movimento regolare, lineare e rapido dei prodotti di lavoro da un passo al successivo in un value stream rilevante.
frequenza di deploy	Frequenza dei rilasci di software in produzione.
idempotenza	Proprietà di un'operazione che produce lo stesso risultato sia che venga eseguita una sola volta sia che venga eseguita più volte.
infrastructure as code (IaC)	Pratica consistente nel gestire ed effettuare il provisioning dell'infrastruttura IT utilizzando file di configurazione leggibili dalla macchina e applicandovi le pratiche dello sviluppo software.
merge	Pratica consistente nell'integrare i cambiamenti di codice da un ramo a un altro ramo.
observability	Capacità di misurare lo stato interno di un sistema esaminandone l'output.
over-provisioning	Pratica consistente nell'allocare a un componente o a un sistema più risorse di elaborazione del necessario, per migliorarne prestazioni o affidabilità.
Personally Identifiable Information (PII)	Qualsiasi informazione collegata a una specifica persona e utilizzabile per risalire alla sua identità.
pipeline di CI/CD	Serie automatizzata di passi per rilasciare una nuova versione del software.
pull request (PR)	Meccanismo che consente di notificare ai membri del team i cambiamenti apportati a una codebase, proponendo che tali cambiamenti siano esaminati, discussi e integrati.
release management	Processo di pianificazione e schedulazione dei release, comprendente il testing e il deployment del software.
release on demand	Pratica consistente nel rilasciare nuove funzionalità sulla base delle esigenze degli stakeholder, disaccoppiando deployment e release.
Service Level Agreement (SLA)	Accordo con un cliente in base al quale un Service Level Objective sarà rispettato per un determinato periodo.
Service Level Indicator (SLI)	Misura quantificabile dell'affidabilità di un servizio.
Service Level Objective (SLO)	Valore obiettivo di affidabilità per un Service Level Indicator.
Single Source of Truth (SSOT)	Pratica di normalizzazione dei dati basata sull'uso di un'unica fonte per uno specifico elemento di dati.
Site Reliability Engineering (SRE)	Approccio che applica pratiche e principi dello sviluppo software all'infrastruttura e alle operations, bilanciando l'affidabilità del servizio

Quality in DevOps (CT-QDO) Syllabus

Termine	Definizione
	con il ritmo di sviluppo delle nuove funzionalità mediante cicli di feedback automatizzati.
Software Bill of Materials (SBOM)	Registrazione leggibile dalla macchina che dettaglia i componenti software e le loro relazioni di supply chain, assicurando trasparenza, auditability e tracciabilità.
Source Code Management (SCM)	Pratica consistente nel tracciare, gestire e archiviare i cambiamenti al software, abilitando version control, sviluppo parallelo e risoluzione dei conflitti tra i contributori.
strategia di branch	Insieme di regole per la gestione dei rami nei version control system.
strategia di rilascio	Strategia volta a determinare come rilasciare il software in produzione, come renderlo disponibile agli utenti e con quale frequenza.
team DevOps cross-funzionale	Gruppo di persone con insiemi di conoscenze, competenze e capacità differenti ma parzialmente sovrapposti, che lavorano insieme verso un obiettivo comune per creare ed esercire un sistema.
telemetria	Raccolta, aggregazione e analisi automatizzate di metriche provenienti da applicazioni e infrastruttura in produzione, per monitorare lo stato di salute del sistema e supportare l'individuazione dei difetti.
tempo di recupero da un deploy fallito	Durata necessaria a ripristinare il normale esercizio dopo un deployment fallito.
under-provisioning	Pratica consistente nell'allocare a un componente o a un sistema meno risorse di elaborazione del necessario, con conseguente degrado delle prestazioni e potenziali failure di sistema.
value stream	Sequenza end-to-end di attività necessarie a consegnare un prodotto o un servizio a un cliente, compreso il flusso di informazioni e di materiali.
version control	Processo che registra nel tempo i cambiamenti agli elementi di configurazione.
version control system (VCS)	Strumento software che automatizza il version control.
virtualizzazione	Tecnica che consente l'erogazione di servizi virtuali rilasciati, resi accessibili e gestiti da remoto.
wall of confusion	Mancanza di comunicazione tra i team di sviluppo e quelli operativi all'interno di un'organizzazione.

6 Trademark

DASA® è un marchio registrato di DevOps Agile Skills Association.

DORA® è un marchio registrato di DevOps Research and Assessment.

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

TMMi® è un marchio registrato di TMMi Foundation.

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

Gli specifici obiettivi di apprendimento applicabili a questo syllabus sono riportati all'inizio di ogni capitolo. Ogni argomento del syllabus sarà oggetto d'esame secondo il relativo obiettivo di apprendimento.

Gli obiettivi di apprendimento iniziano con un verbo d'azione corrispondente al loro livello cognitivo di conoscenza, come indicato di seguito.

Livello 1: Ricordare (K1)

Il candidato ricorda, riconosce e richiama un termine o un concetto.

Verbi d'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 di affermazioni relative all'argomento ed è in grado di riassumere, confrontare, classificare e fornire esempi relativi al concetto di testing.

Verbi d'azione: Classificare, confrontare, differenziare, distinguere, spiegare, fornire esempi, inferire, riassumere

Esempi	Note
Classificare gli strumenti di test in base alla loro finalità e alle attività di test che supportano.	
Confrontare i diversi livelli di test.	Può essere usato per cercare somiglianze, differenze o entrambe.
Differenziare il testing dal debugging.	Cerca le differenze tra 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.	
Inferire la root cause dei difetti da un dato profilo di failure.	
Riassumere le attività del processo di review dei prodotti di lavoro.	

Livello 3: Applicare (K3)

Il candidato è in grado di svolgere una procedura di fronte a un'attività nota, oppure di selezionare la procedura corretta e applicarla a un dato contesto.

Verbi d'azione: Applicare, implementare, predisporre, utilizzare

Quality in DevOps (CT-QDO) Syllabus

Esempi	Note
Applicare l'analisi dei valori limite per derivare test case da requisiti dati.	Dovrebbe fare riferimento a una procedura/tecnica/processo ecc.
Implementare metodi di raccolta delle metriche a supporto dei requisiti tecnici e gestionali.	
Predisporre test di installabilità per app mobile.	
Utilizzare la tracciabilità per monitorare l'avanzamento del test rispetto a completezza e consistenza con gli obiettivi del test, la strategia di test e il test plan.	Potrebbe essere usato in un LO che richiede al candidato di saper utilizzare una tecnica o una procedura. Simile ad «applicare».

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

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

Questa sezione riporta la tracciabilità tra i Business Outcome e gli Obiettivi di Apprendimento di Quality in DevOps.

Business Outcome: Quality in DevOps		QDO-BO1	QDO-BO2	QDO-BO3	QDO-BO4	QDO-BO5	QDO-BO6	QDO-BO7	QDO-BO8	QDO-BO9
QDO-BO1	Spiegare Come la Quality Assurance è Supportata dai Concetti DevOps e Vi Contribuisce	6								
QDO-BO2	Comprendere i Concetti dell'Implementazione di DevOps in un'Organizzazione		3							
QDO-BO3	Contribuire a Tutte le Fasi del Value Stream a Supporto del Quality Engineering			4						
QDO-BO4	Contribuire all'Implementazione del Loop DevOps				6					
QDO-BO5	Descrivere Come l'Automazione Supporta la Quality Assurance in DevOps					6				
QDO-BO6	Implementare il Testing Automatizzato nei Team DevOps						3			
QDO-BO7	Implementare il Testing Manuale nei Team DevOps							4		
QDO-BO8	Selezionare gli Strumenti a Supporto di DevOps all'Interno dell'Organizzazione								2	
QDO-BO9	Comprendere Tecnologie e Pratiche a Supporto della Qualità in DevOps									9

Business Outcome: Quality in DevOps		QDO-BO1	QDO-BO2	QDO-BO3	QDO-BO4	QDO-BO5	QDO-BO6	QDO-BO7	QDO-BO8	QDO-BO9
Numero LO	Obiettivo di Apprendimento (Livello K)									
1	Fondamenti di DevOps - 140 minuti									
1.1	Spiegare Come la Quality Assurance è Supportata dai Concetti DevOps e Vi Contribuisce									
QDO-1.1.1	Richiamare i concetti chiave di DevOps (K1)	×								
QDO-1.1.2	Richiamare i concetti relativi al wall of confusion in DevOps (K1)	×								
QDO-1.1.3	Spiegare le metriche DORA di delivery performance e operational performance (K2)	×								
QDO-1.1.4	Differenziare gli elementi di CALMS in termini di Quality Assurance (K2)	×								
QDO-1.1.5	Spiegare come la Quality Assurance supporta i tre modi di DevOps (K2)	×								
QDO-1.1.6	Spiegare i benefici, i rischi e le insidie di DevOps (K2)	×								
1.2	Comprendere i Concetti dell'Implementazione di DevOps in un'Organizzazione									
QDO-1.2.1	Distinguere i ruoli del team DevOps (K2)		×							
QDO-1.2.2	Spiegare il concetto di Site Reliability Engineering (SRE) in relazione a DevOps (K2)		×							
QDO-1.2.3	Distinguere i diversi pattern e anti-pattern dei team DevOps (K2)		×							
2	Quality Assurance (QA) e Testing in DevOps - 360 minuti									
2.1	Contribuire a Tutte le Fasi del Value Stream a Supporto del Quality Engineering									
QDO-2.1.1	Implementare attività di Quality Assurance in un contesto DevOps (K3)			×						
QDO-2.1.2	Confrontare gli obiettivi del test che supportano DevOps con i modelli di sviluppo sequenziali e con lo sviluppo software Agile (K2)			×						
QDO-2.1.3	Spiegare il continuous testing in un contesto DevOps (K2)			×						
QDO-2.1.4	Spiegare come le pull request supportano la qualità (K2)			×						

Business Outcome: Quality in DevOps		QDO-BO1	QDO-BO2	QDO-BO3	QDO-BO4	QDO-BO5	QDO-BO6	QDO-BO7	QDO-BO8	QDO-BO9
Numero LO	Obiettivo di Apprendimento (Livello K)									
2.2	Contribuire all'Implementazione del Loop DevOps									
QDO-2.2.1	Spiegare Quality Assurance e testing nella continuous discovery e le relative pratiche (K2)				×					
QDO-2.2.2	Spiegare Quality Assurance e testing nella continuous integration e le relative pratiche (K2)				×					
QDO-2.2.3	Spiegare Quality Assurance e testing nella continuous delivery e le relative pratiche (K2)				×					
QDO-2.2.4	Spiegare Quality Assurance e testing nel continuous deployment e le relative pratiche (K2)				×					
QDO-2.2.5	Spiegare Quality Assurance e testing nel release on demand e le relative pratiche (K2)				×					
QDO-2.2.6	Applicare le conoscenze di Quality Assurance e testing per implementare una pipeline di CI/CD (K3)				×					
3	Attività Automatizzate e Manuali in DevOps - 510 minuti									
3.1	Descrivere Come l'Automazione Supporta la Quality Assurance in DevOps									
QDO-3.1.1	Spiegare come una Single Source of Truth per il testware supporta la Quality Assurance (K2)					×				
QDO-3.1.2	Spiegare come l'automazione supporta la tracciabilità tra la base di test e il testware (K2)					×				
QDO-3.1.3	Implementare pratiche uniformi di reporting della qualità nella pipeline di CI/CD con informazioni provenienti sia dal testing automatizzato sia dal testing manuale (K3)					×				
QDO-3.1.4	Spiegare i benefici dell'automazione della gestione dei dati di test (K2)					×				
QDO-3.1.5	Spiegare i benefici dell'analisi statistica dei risultati dei test su lunghi periodi di tempo (K2)					×				

Business Outcome: Quality in DevOps		QDO-BO1	QDO-BO2	QDO-BO3	QDO-BO4	QDO-BO5	QDO-BO6	QDO-BO7	QDO-BO8	QDO-BO9
Numero LO	Obiettivo di Apprendimento (Livello K)									
QDO-3.1.6	Spiegare i benefici di una gestione degli ambienti di test standardizzata, controllata e automatizzata (K2)					×				
3.2	Implementare il Testing Automatizzato nei Team DevOps									
QDO-3.2.1	Spiegare come il regression testing si integra nelle varie attività della pipeline di CI/CD (K2)						×			
QDO-3.2.2	Predisporre il testing delle API (K3)						×			
QDO-3.2.3	Confrontare l'estensione della test automation in DevOps rispetto allo sviluppo software Agile e ai modelli di sviluppo sequenziali (K2)						×			
3.3	Implementare il Testing Manuale nei Team DevOps									
QDO-3.3.1	Fornire esempi di testing manuale per un'organizzazione DevOps (K2)							×		
QDO-3.3.2	Spiegare come il testing esplorativo può funzionare con una pipeline di CI/CD (K2)							×		
QDO-3.3.3	Spiegare come il crowd testing può supportare la cultura del feedback (K2)							×		
QDO-3.3.4	Applicare eventi di quality hunting a supporto della cultura dell'apprendimento (K3)							×		
4	Strumenti e Pratiche in DevOps - 200 minuti									
4.1	Selezionare gli Strumenti a Supporto di DevOps all'Interno dell'Organizzazione									
QDO-4.1.1	Richiamare le capability degli strumenti in DevOps (K1)								×	
QDO-4.1.2	Spiegare gli strumenti a supporto della Quality Assurance e del testing in DevOps (K2)								×	
4.2	Comprendere Tecnologie e Pratiche a Supporto della Qualità in DevOps									
QDO-4.2.1	Richiamare le attività non di testing all'interno della pipeline di CI/CD (K1)									×
QDO-4.2.2	Spiegare le diverse strategie di rilascio principali in DevOps (K2)									×
QDO-4.2.3	Riassumere i concetti di infrastructure as code (K2)									×

v1.0 Versione di Rilascio

Business Outcome: Quality in DevOps		QDO-BO1	QDO-BO2	QDO-BO3	QDO-BO4	QDO-BO5	QDO-BO6	QDO-BO7	QDO-BO8	QDO-BO9
Numero LO	Obiettivo di Apprendimento (Livello K)									
QDO-4.2.4	Richiamare i feature toggle come modo per separare il deployment dal release (K1)									×
QDO-4.2.5	Richiamare le strategie di branch (K1)									×
QDO-4.2.6	Richiamare il chaos engineering (K1)									×
QDO-4.2.7	Richiamare telemetria e observability (K1)									×
QDO-4.2.8	Richiamare la Software Bill of Materials a supporto della gestione della configurazione (K1)									×
QDO-4.2.9	Richiamare la containerizzazione (K1)									×

9 Appendice C – Note di Rilascio

Questo syllabus è il primo rilascio del syllabus Certified Tester Quality in DevOps (CT-QDO) dell'International Software Testing Qualification Board.

10 Riferimenti

Standard

Systems and software engineering (25010): Systems and software Quality Requirements and Evaluation (SQuaRE) — Product quality model, 2023. 2023-01. Standard. International Organization for Standardization.

Documenti ISTQB®

ISTQB® CT-AT: Agile Tester Syllabus, 2014. Version 1.1.

ISTQB® CT-ATLaS: Agile Test Leadership at Scale Syllabus, 2023. Version 2.0.

ISTQB® CT-ATT: Agile Technical Tester Syllabus, 2019. Version 1.1.

ISTQB® CT-MAT: Mobile Application Testing Syllabus, 2019. Version 2019.

ISTQB® CT-PT: Performance Testing Syllabus, 2018. Version 2018.

ISTQB® CT-SEC: Security Testing Syllabus, 2016. Version 1.0.

ISTQB® CT-TAS: Test Automation Strategy Syllabus, 2024. Version 1.0.

ISTQB® CTAL-TAE: Advanced Level Test Automation Engineer Syllabus, 2024. Version 2.0.

ISTQB® CTAL-TM: Advanced Level Test Management Syllabus, 2024. Version 3.0.

ISTQB® CTFL: Foundation Level Syllabus, 2023. Version 4.0.

ISTQB® Exam Structures and Rules, 2025. Version 1.2.

ISTQB® Generic Accreditation Guidelines, 2024. Version 2.4.

Libri

ADZIC, Gojko, 2011. *Specification by Example: How Successful Teams Deliver the Right Software*. Shelter Island, NY: Manning Publications. isbn 9781935182553.

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.

AXELOS, 2019. *ITIL Foundation: ITIL 4 Edition*. London, UK: TSO (The Stationery Office). ISBN: 978-0113316076.

BECK, Kent, 2000. *Extreme Programming Explained: Embrace Change*. Addison-Wesley.

BEYER, Betsy; JONES, Chris; PETOFF, Jennifer; MURPHY, Niall Richard, 2016. *Site Reliability Engineering: How Google Runs Production Systems*. 1st. O'Reilly Media, Inc. isbn 149192912X.

CLOKIE, K., 2017. *A Practical Guide to Testing in DevOps*. Leanpub. Available also from: <https://leanpub.com/testingindevops>.

Quality in DevOps (CT-QDO) Syllabus

- CRISPIN, L.; GREGORY, J., 2008. *Agile Testing: A Practical Guide for Testers and Agile Teams*. Pearson Education. Addison-Wesley Signature Series (Cohn). isbn 9780321616937. Available also from: https://books.google.hu/books?id=68_lhPvoKS8C.
- HENDRICKSON, E., 2013. *Explore It!: Reduce Risk and Increase Confidence with Exploratory Testing*. Pragmatic Bookshelf. The Pragmatic Bookshelf. isbn 9781937785024. Available also from: <https://books.google.nl/books?id=jxqpMQEACAAJ>.
- HUMBLE, J.; FARLEY, D., 2010. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Pearson Education. Addison-Wesley Signature Series (Fowler). isbn 9780321670229. Available also from: <https://books.google.hu/books?id=6ADDuzere-YC>.
- HUMBLE, J.; MOLESKY, J.; O'REILLY, B., 2020. *Lean Enterprise*. O'Reilly Media. isbn 9781492092223. Available also from: <https://books.google.hu/books?id=BmbyDwAAQBAJ>.
- KIM, G.; BEHR, K.; SPAFFORD, G., 2018. *The Phoenix Project: A Novel about IT, DevOps, and Helping Your Business Win*. IT Revolution Press. isbn 9781942788300. Available also from: <https://books.google.hu/books?id=H6x-DwAAQBAJ>.
- KIM, G.; HUMBLE, J.; DEBOIS, P.; WILLIS, J., 2016. *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution Press. ITpro collection. isbn 9781942788072. Available also from: <https://books.google.hu/books?id=ui8hDgAAQBAJ>.
- LARMAN, Craig; VODDE, Bas, 2016. *Large-scale scrum: More with LeSS*. Addison-Wesley Professional.
- MARSELIS, R.; GEURTS, D.; RUIGROK, W.; VEENENDAAL, B. van, 2020. *Quality for DevOps teams*. Sogeti Nederland B.V. isbn 9789075414905. Available also from: <https://books.google.hu/books?id=ICHXDwAAQBAJ>.
- RUIGROK, W.; EGELMEERS, J.; MARSELIS, R., 2025. *Amplified Quality Engineering*. Sogeti Nederland B.V. isbn 9789075414929.

Articoli

- DEBOIS, Patrick, 2011. Devops: A software revolution in the making. *Cutter IT Journal*. Vol. 24, no. 8, pp. 3–5.

Pagine Web

- ALLIANCE, Agile, 2026. *Agile Glossary and Terminology* [online]. Agile Alliance, 2026-01-31 [visited on 2026-01-31]. Available from: <https://agilealliance.org/agile101/agile-glossary/>.
- AMAZON, 2024. *What is DevOps?* [online]. Amazon, 2024-10-15 [visited on 2024-10-15]. Available from: <https://aws.amazon.com/devops/what-is-devops/>.
- DASA, 2019. *Embracing digital disruption by adopting DevOps practices* [online]. 2019-01-01. [visited on 2024-07-09]. Available from: <https://www.dasa.org/wp-content/uploads/DASA-White-Paper-1-Embracing-Digital-Disruption-1.pdf>.
- DASA, 2024. *Challenges in Breaking The Wall of Confusion: DASA DevOps Fundamentals, Knowledge Bytes* [online]. DASA, 2024-07-05 [visited on 2024-08-09]. Available from: <https://www.dasa.org/blog/challenges-in-breaking-the-wall-of-confusion/>.

v1.0 Versione di Rilascio

Quality in DevOps (CT-QDO) Syllabus

- DIGITAL.AI, 2023. *DevSecOps Tools Periodic Table* [online]. [visited on 2024-11-13]. Available from: <https://digital.ai/learn/devsecops-periodic-table/>.
- DORA, 2022. *2022 Accelerate State of DevOps Report* [online]. DORA, 2022-12-01 [visited on 2024-08-12]. Available from: <https://dora.dev/research/2022/dora-report/2022-dora-accelerate-state-of-devops-report.pdf>.
- DORA, 2024. *DORA's software delivery metrics: the four keys* [online]. DORA, 2024-05-30 [visited on 2024-08-12]. Available from: <https://dora.dev/guides/dora-metrics-four-keys/>.
- FELLOWS, Matt, 2022. *Pact - Getting started* [online]. 2022-08-30. [visited on 2025-06-24]. Available from: <https://docs.pact.io/>.
- FOWLER, Martin, 2017. *Feature Toggles* [online]. 2017-10-09. [visited on 2017-10-09]. Available from: <https://martinfowler.com/articles/feature-toggles.html>.
- ISTQB, 2025. *Glossary* [online]. ISTQB, 2025-05-15 [visited on 2025-05-15]. Available from: <https://glossary.istqb.org/>.
- JANET GREGORY, Lisa Crispin, 2023. *Holistic Testing* [online]. Gregory/Crispin [visited on 2024-09-05]. Available from: <https://agiletestingfellow.com/>.
- KUNDU, Sourav, 2019. *Idempotency in Infrastructure as Code* [online]. [visited on 2025-01-24]. Available from: <https://skundunotes.com/2019/04/19/idempotency-in-infrastructure-as-code/>.
- MATTHEW SKELTON, Manuel Pais, 2023. *What Team Structure is Right for DevOps to Flourish?* [online]. Skelton/Pais, 2023-03-21 [visited on 2024-09-03]. Available from: <https://web.devopstopologies.com/>.
- MOUSTAKIS, Ioannis, 2024. *Idempotency in Infrastructure as Code* [online]. 2024-10-31. [visited on 2025-01-26]. Available from: <https://spacelift.io/blog/what-is-configuration-drift>.
- NETFLIX, 2025. *Chaos Engineering* [online]. netflixtechblog, 2025-01-05 [visited on 2025-01-05]. Available from: <https://netflixtechblog.com/tagged/chaos-engineering>.
- RANGANATHAN, Rahul, 2023. *Building Resilient Systems with Chaos Engineering* [online]. Medium, 2023-07-27 [visited on 2025-01-05]. Available from: <https://medium.com/google-cloud/building-resilient-systems-with-chaos-engineering-91f5b6adc972>.
- Scaled Agile Framework: CALMR*, 2023a [online]. © Scaled Agile, Inc., 2023-03-14 [visited on 2024-08-12]. Available from: <https://scaledagileframework.com/calmr/>.
- Scaled Agile Framework: Customer Centricity*, 2023b [online]. © Scaled Agile, Inc., 2023-03-14 [visited on 2024-10-31]. Available from: <https://scaledagileframework.com/customer-centricity/>.
- Scaled Agile Framework: Design Thinking*, 2023c [online]. © Scaled Agile, Inc., 2023-03-14 [visited on 2024-10-31]. Available from: <https://scaledagileframework.com/design-thinking/>.
- TORRES, Teresa, 2024. *Blog: Continuous Discovery Framework* [online]. <https://userpilot.com/>, 2024-09-16 [visited on 2024-10-31]. Available from: <https://userpilot.com/blog/continuous-discovery-framework-teresa-torres/#What-is-continuous-discovery?>.
- VEENENDAAL, Erik van, 2025. *TMMi® in the DevOps world* [online]. [visited on 2025]. Available from: <https://www.tmmi.org/download/tmmi-in-devops/>.

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 ritenuto responsabile qualora non siano più disponibili.

11 Ulteriori Letture

- ARIOLA, W.; DUNLOP, C., 2014. *Continuous Testing*. CreateSpace Independent Publishing Platform. isbn 9781494859756. Available also from: <https://books.google.hu/books?id=MatcngEACAAJ>.
- BELMONT, J.M., 2018. *Hands-On Continuous Integration and Delivery: Build and release quality software at scale with Jenkins, Travis CI, and CircleCI*. Packt Publishing. isbn 9781789133073. Available also from: <https://books.google.hu/books?id=hh9sDwAAQBAJ>.
- BLANK-EDELMAN, D.N., 2018. *Seeking SRE: Conversations About Running Production Systems at Scale*. O'Reilly Media. isbn 9781491978818. Available also from: <https://books.google.hu/books?id=tmhqDwAAQBAJ>.
- BRYKCZYNSKI, Bill, 1992. A survey of software inspection checklists. *ACM SIGSOFT Software Engineering Notes*. Vol. 24, no. 1, pp. 82–89.
- DUNLOP, C.; PLATZ, W., 2019. *Enterprise Continuous Testing: Transforming Testing for Agile and DevOps*. Independently Published. isbn 9781699022948. Available also from: <https://books.google.hu/books?id=ms1ZywEACAAJ>.
- DUVALL, P.M.; MATYAS, S.; GLOVER, A., 2007. *Continuous Integration: Improving Software Quality and Reducing Risk*. Pearson Education. Addison-Wesley Signature Series. isbn 9780321630148. Available also from: <https://books.google.hu/books?id=PV9qfEdv9L0C>.
- FORSGREN, N.; HUMBLE, J.; KIM, G., 2018. *Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations*. IT Revolution Press. isbn 9781942788355. Available also from: <https://books.google.hu/books?id=Kax-DwAAQBAJ>.
- GREGORY, J.; CRISPIN, L., 2019. *Agile Testing Condensed: A Brief Introduction*. Leanpub. isbn 9781999220518. Available also from: <https://books.google.hu/books?id=yP-yzwEACAAJ>.
- KIM, G., 2019. *The Unicorn Project: A Novel about Developers, Digital Disruption, and Thriving in the Age of Data*. IT Revolution Press. isbn 9781942788775. Available also from: <https://books.google.hu/books?id=kNSSDwAAQBAJ>.
- KINSBRUNER, E., 2018. *Continuous Testing for DevOps Professionals: A Practical Guide From Industry Experts*. CreateSpace Independent Publishing Platform. isbn 9781727132175. Available also from: https://books.google.fi/books?id=_I_ruwEACAAJ.
- MURPHY, N.R.; BEYER, B.; JONES, C.; PETOFF, J., 2016. *Site Reliability Engineering: How Google Runs Production Systems*. O'Reilly Media. isbn 9781491951170. Available also from: <https://books.google.hu/books?id=tYrPCwAAQBAJ>.
- NOVOTNÝ, Vít, 2017. Using Markdown inside TEX documents. *TUGboat*. Vol. 38, no. 2, pp. 214–217.
- ROSSEL, S., 2017. *Continuous Integration, Delivery, and Deployment: Reliable and faster software releases with automating builds, tests, and deployment*. Packt Publishing. isbn 9781787284180. Available also from: <https://books.google.hu/books?id=1xhKDwAAQBAJ>.
- SCHEAFFER, J.; RAVICHANDRAN, A.; MARTINS, A., 2018. *The Kitty Hawk Venture: A Novel About Continuous Testing in DevOps to Support Continuous Delivery and Business Success*. Apress. isbn 9781484236611. Available also from: <https://books.google.hu/books?id=asRIDwAAQBAJ>.

Quality in DevOps (CT-QDO) Syllabus

- VADAPALLI, S., 2018. *DevOps: Continuous Delivery, Integration, and Deployment with DevOps: Dive into the core DevOps strategies*. Packt Publishing. isbn 9781789131253. Available also from: <https://books.google.hu/books?id=N5RRDwAAQBAJ>.
- VEENENDAAL, Erik P.W.M. van; WELLS, B., 2012. *Test Maturity Model Integration TMMi: Guidelines for Test Process Improvement*. Tutein Nolthenius. isbn 9789490986100. Available also from: <https://books.google.fi/books?id=6y8QnQEACAAJ>.
- ZHAN, C.; ZHAN, Z., 2021. *Practical Continuous Testing: Make Agile/DevOps Real*. CreateSpace Independent Publishing Platform. isbn 9781507742112. Available also from: <https://books.google.hu/books?id=6QG1zgEACAAJ>.

12 Appendice D – Indice Analitico

Acceptance Test-Driven Development; 22
analisi delle dipendenze; 32
analisi statica; 15
analisi statistica; 34
artefatto di build; 34
Behavior-Driven Development; 22
blue-green deployment; 22
branch; 48
CALMS; 12
canary release; 22
change fail percentage; 12
change lead time; 12
chaos engineering; 15
ciclo di feedback; 12
code smell; 51
containerizzazione; 32
continuous delivery; 7
continuous deployment; 14
continuous discovery; 14
continuous integration; 13
continuous learning and experimentation; 16
continuous monitoring; 22
continuous testing; 22
crowd testing; 34
dark launch; 22
dati di test; 28
definition of done; 28
definition of ready; 28
DevOps; 7
fault injection; 48
feature branch; 56
feature toggle; 32
file binario; 34
flusso; 12
frequenza di deploy; 12
gestione dei dati di test; 34
hotfix; 15
idempotenza; 54
infrastructure as code; 17
merge; 26
metriche DORA; 14
modello di sviluppo sequenziale; 22
observability; 19
over-provisioning; 41
performance testing; 39
Personally Identifiable Information; 34
pipeline di CI/CD; 14
piramide di test; 24
politica della qualità; 22
politica di test; 22
pull request; 19
quadranti del testing; 24
Quality Assurance; 7
quality engineering; 7
quality gate; 26
quality hunting; 15
quality report; 34
regression testing; 16
release management; 48
release on demand; 7
reporting della qualità; 34
risultato del test; 34
Service Level Agreement; 12
Service Level Indicator; 12
Service Level Objective; 12
shift-left; 42
Single Source of Truth; 34
Site Reliability Engineering; 12
Software Bill of Materials; 48
Source Code Management; 48
strategia di branch; 48
strategia di rilascio; 29
sviluppo software Agile; 7
team DevOps cross-funzionale; 12
telemetria; 19
tempo di recupero da un deploy fallito; 12
test automation; 8
testing A/B; 22
testing automatizzato; 11
testing delle API; 34
testing di contratti; 34
testing di sicurezza; 42
testing esplorativo; 25
testing olistico; 22
testing statico; 22
tolleranza ai guasti; 48

Quality in DevOps (CT-QDO) Syllabus

tracciabilità; 26
trunk-based development; 55
under-provisioning; 41
unit testing; 44
value stream; 11
value stream mapping; 31

version control; 26
version control system; 26
virtualizzazione; 32
walking skeleton; 32
wall of confusion; 12