

TMMi[®] nel mondo DevOps (v1.1)

#SuccessWithTMMi

EDITOR: ERIK VAN VEENENDAAL

TMMi è concepito per essere indipendente dal ciclo di vita. Questo documento spiega come il modello di miglioramento di test TMMi possa essere utilizzato e applicato in modo vantaggioso in un contesto DevOps. Le Aree di Processo TMMi di livello 2 e di livello 3 e i relativi Obiettivi sono trattati uno per uno. Viene indicato come queste si relazionano a DevOps e quale forma assumono tipicamente le relative pratiche. Se di una pratica non è prevista l'esecuzione in un contesto DevOps, in quanto priva di valore aggiunto, ciò viene esplicitamente dichiarato.

Si precisa che questo documento non è pensato come un syllabus completo sul testing in ambito DevOps, ma piuttosto come un documento che mostra “semplicemente” come dovrebbero essere interpretati gli obiettivi del TMMi in un contesto DevOps, offrendo spunti e indicazioni per ulteriori approfondimenti. TMMi in the DevOps world non è inoltre inteso come una descrizione che sostituisca il modello TMMi completo. TMMi in the DevOps world dovrebbe essere sempre utilizzato congiuntamente al documento di riferimento del modello TMMi.

© TMMi Foundation 2011-25

Tutti i diritti riservati. Nessuna parte di questa pubblicazione può essere prestata, venduta, trasferita, riprodotta o trasmessa, nella sua interezza o in parte, in qualsiasi forma o con qualsiasi mezzo senza previa autorizzazione della TMMi Foundation, salvo quanto descritto nella documentazione di licenza associata. Qualora qualsiasi forma di copia sia consentita ai sensi dei termini della documentazione di licenza associata, ciò è subordinato al vincolo che questo avviso venga riprodotto in tutte le copie.

Le parole che riteniamo possano costituire marchi sono state designate come tali. Tuttavia, né la presenza né l'assenza di tale designazione dovrebbero essere considerate come influenti sullo status legale di qualsiasi marchio.

TMMi è un marchio registrato della TMMi Foundation (UK).

Hanno contribuito:

Katalin Balla (Hungary)

Suresh Chandra Bose (USA)

Kari Kakkonen (Finland)

Mattijs Kemmink (The Netherlands)

Rik Marselis (The Netherlands)

Szilard Szell (Hungary)

Erik van Veenendaal (The Netherlands)

Per la traduzione italiana:

Raffaele Bozzini

Giorgio Pisani

Salvatore Reale

Revisioni

Questa sezione riassume le principali revisioni tra le varie versioni del documento.

Questa sezione è fornita solo a titolo informativo.

Versione	Note di revisione
V1.0	Versione iniziale del documento "TMMi nel mondo DevOps".
V1.1	È stata aggiunta l'interpretazione delle aree di processo, degli obiettivi e delle pratiche del TMMi di livello 3 in un contesto DevOps. È stato inoltre aggiunto un glossario per i termini specifici di DevOps.

Indice dei contenuti

Hanno contribuito:	2
Revisioni	3
Indice dei contenuti	4
1 TMMi® nel mondo DevOps	6
1.1 Introduzione	6
1.2 DevOps e DevSecOps	6
1.3 Test Maturity Model integration (TMMi)	7
1.4 Testing in un contesto DevOps	8
1.4.1 Le sfide del testing in un contesto DevOps	8
1.4.2 TMMi e DevOps	10
1.4.3 Relazione con il syllabus DevOps di ISTQB	11
2 Livello 2 TMMi - Managed	12
2.1 Area di Processo 2.1 Test Policy and Strategy	12
2.1.1 SG1 Stabilire una politica di test	12
2.1.2 SG2 Stabilire una strategia di test	14
2.1.3 SG3 Stabilire i performance indicator dei test	17
2.2 Area di Processo 2.2 Test Planning	18
2.2.1 SG1 Eseguire un assessment del rischio di prodotto	18
2.2.2 SG1 Stabilire un approccio di test	19
2.2.3 SG3 Stabilire le stime di test	21
2.2.4 SG4 Sviluppare un test plan	22
2.2.5 SG5 Ottenere commitment per il test plan	23
2.3 Area di Processo 2.3 Test Monitoring and Control	23
2.3.1 SG1 Monitorare l'avanzamento dei test rispetto al piano	24
2.3.2 SG2 Monitorare la qualità del prodotto rispetto a piano e aspettative	25
2.3.3 SG3 Gestire le azioni correttive fino alla chiusura	27
2.4 Area di Processo 2.4 Test Design and Execution	27
2.4.1 SG1 Eseguire l'analisi e la progettazione dei test utilizzando tecniche di progettazione dei test	28
2.4.2 SG2 Eseguire l'implementazione dei test	29
2.4.3 SG3 Eseguire l'esecuzione dei test	30
2.4.4 SG4 Gestire gli incidenti di test fino alla chiusura	31
2.5 Area di Processo 2.5 Test Environment	32
2.5.1 SG1 Sviluppare i requisiti dell'ambiente di test	32
2.5.2 SG2 Eseguire l'implementazione dell'ambiente di test	33
2.5.3 SG3 Gestire e controllare gli ambienti di test	33
3 Livello 3 TMMi - Defined	35

3.1	Area di Processo 3.1 Test Organization	35
3.1.1	SG1 Istituire un'Organizzazione di Test.....	35
3.1.2	SG2 Istituire le Funzioni di Test per gli Specialisti del Test	36
3.1.3	SG3 Definire i Percorsi di Carriera nel Test.....	36
3.1.4	SG4 Determinare, Pianificare e Implementare i Miglioramenti del Processo di Test.....	37
3.1.5	SG5 Distribuire il Processo di Test Organizzativo e Incorporare le Lesson Learned	38
3.2	Area di Processo 3.2 Test Training Program	38
3.2.1	SG1 Istituire una Capacità Organizzativa di Formazione sul Test	39
3.2.2	SG2 Fornire la Formazione sul Test Necessaria	39
3.3	Area di Processo 3.3 Test Lifecycle and Integration	40
3.3.1	SG1 Istituire gli Asset Organizzativi di Processo di Test	40
3.3.2	SG2 Integrare i Modelli di Ciclo di Vita del Test con i Modelli di Sviluppo	42
3.3.3	SG3 Istituire un Master Test Plan	42
3.4	Area di Processo 3.4 Non-Functional Testing.....	43
3.4.1	SG1 Eseguire una Valutazione dei Rischi di Prodotto Non Funzionali	43
3.4.2	SG2 Definire un Approccio di Test Non Funzionale	44
3.4.3	SG3 Eseguire l'Analisi e la Progettazione dei Test Non Funzionali.....	45
3.4.4	SG4 Eseguire l'Implementazione dei Test Non Funzionali	46
3.4.5	SG5 Svolgere l'Esecuzione dei Test Non Funzionali.....	46
3.5	Area di Processo 3.5 Peer Reviews	47
3.5.1	SG1 Definire un Approccio di Peer Review	47
3.5.2	SG2 Eseguire le Peer Review	49
	Glossario dei termini specifici DevOps.....	51
	Riferimenti.....	53

1 TMMi® nel mondo DevOps

1.1 Introduzione

Questo documento spiega come la combinazione di un approccio DevOps per lo sviluppo del software con il modello di miglioramento del processo di test TMMi sia un percorso possibile per raggiungere gli obiettivi di business. Spiega come TMMi possa essere utilizzato e applicato in modo vantaggioso in un contesto DevOps. Vengono fornite alternative comprovate basate su DevOps agli approcci tradizionali del testing che implementano le pratiche TMMi. Questo documento spiega come TMMi può aiutare le organizzazioni DevOps, indipendentemente dalla maturità dell'organizzazione nel loro percorso DevOps, fornendo promemoria delle pratiche di testing critiche che spesso perdono visibilità con la crescita delle organizzazioni e l'aumento della pressione sui progetti. Ogni Area di Processo TMMi e i suoi Obiettivi Specifici sono discussi uno per uno. Viene illustrato il loro rapporto con DevOps e l'aspetto tipico delle pratiche associate. Se non si prevede che una pratica venga eseguita in un contesto DevOps, poiché non ha alcun valore aggiunto, viene esplicitamente dichiarato.

Nel frenetico mondo DevOps di oggi, in cui la continuous integration, la continuous delivery e l'automazione guidano lo sviluppo del software, garantire la qualità in ogni fase è fondamentale. Il Test Maturity Model Integration (TMMi) fornisce un framework strutturato per valutare e migliorare le pratiche del testing. Poiché i team DevOps si sforzano di ottenere release rapide senza compromettere la qualità, l'integrazione del TMMi offre una roadmap per migliorare la maturità del testing, allinearsi agli obiettivi di business e ottenere risultati. Questa sinergia tra TMMi e DevOps crea un approccio equilibrato per accelerare lo sviluppo mantenendo elevati standard di qualità e affidabilità.

Questo documento si rivolge a un diverso numero di destinatari, i principali dei quali sono:

- Organizzazioni tradizionali mature che vogliono passare a DevOps mantenendo la maturità dei propri processi
- Organizzazioni meno mature che vogliono iniziare ad applicare TMMi per aumentare la loro maturità nei test e passare a DevOps in parallelo
- Organizzazioni Agile che hanno un processo di testing Agile strutturato, e che ora vogliono passare a DevOps
- Le organizzazioni DevOps che hanno successo e stanno crescendo. Di conseguenza, hanno bisogno di una certa maturità dei processi, ma allo stesso tempo vogliono mantenere i vantaggi dell'Agilità.

Questo documento non è destinato a essere utilizzato indipendentemente dal modello TMMi originale. Ha lo scopo di fornire una guida a coloro che eseguono il miglioramento del processo di test in un ambiente DevOps sull'interpretazione e il significato dei vari Obiettivi e Pratiche TMMi. Questo documento integra il modello TMMi originale e dovrebbe essere utilizzato come documento complementare.

1.2 DevOps e DevSecOps

DevOps è una filosofia e un insieme di pratiche finalizzate a colmare il divario tra i team di sviluppo software e quelli di operations IT. Promuovendo una cultura di collaborazione e responsabilità condivisa, DevOps mira a semplificare lo sviluppo, il deployment e la manutenzione del software. L'idea centrale è abbattere i silos tradizionali tra sviluppo e operations, rendendo la consegna del software più efficiente ed efficace.

Al cuore del DevOps vi è l'enfasi sull'automazione e sui processi continui. L'automazione riveste un ruolo cruciale nella riduzione delle attività manuali, contribuendo così a diminuire gli errori e aumentare la velocità. Ciò include l'automazione dell'integrazione delle modifiche al codice, del testing e del processo di deployment attraverso pratiche di Continuous Integration (CI) e Continuous Deployment (CD).

Un altro aspetto fondamentale della cultura DevOps è la necessità di cambiamento verso un modello basato su autonomia, leadership, collaborazione, sperimentazione frequente, cicli di sviluppo più brevi, maggiore frequenza di deployment e release più affidabili, in stretto allineamento con gli obiettivi di business e i clienti.

L'approccio DevOps porta numerosi benefici, tra cui una riduzione del time-to-market, una migliore collaborazione tra i team, una qualità e affidabilità del software superiori, oltre a una maggiore scalabilità e flessibilità. Permettendo una rilevazione e risoluzione più rapida dei problemi, DevOps riduce anche i tempi di inattività e migliora la resilienza del sistema.

In definitiva, DevOps riguarda tanto il cambiamento culturale quanto la tecnologia e i processi. Richiede un cambio di mentalità in cui i team di sviluppo e operations lavorano insieme verso obiettivi comuni, accolgono il feedback e puntano costantemente al miglioramento. Integrando questi elementi, DevOps aiuta le organizzazioni a consegnare software in modo più efficiente, affidabile e sicuro.

DevSecOps

DevSecOps, abbreviazione di Development, Security, and Operations, è un approccio che integra le pratiche di sicurezza all'interno del processo DevOps. Questa metodologia mira a garantire che la sicurezza sia una responsabilità condivisa lungo l'intero ciclo di vita IT, dallo sviluppo iniziale fino al deployment e oltre.

L'obiettivo di DevSecOps è rilasciare software sicuro in modo più rapido ed efficiente, riducendo il rischio di violazioni della sicurezza.

1.3 Test Maturity Model integration (TMMi)

Il framework TMMi è stato sviluppato dalla TMMi Foundation come linea guida e riferimento per il miglioramento del processo di test, che affronta le problematiche rilevanti per test manager, test engineer, sviluppatori e professionisti della qualità del software. Nel TMMi il concetto di testing è inteso nel suo senso più ampio, per comprendere tutte le attività relative alla qualità del prodotto software.

Il TMMi utilizza il concetto di Livelli di Maturità per la valutazione e il miglioramento del processo. Inoltre, vengono identificate Aree di Processo, Obiettivi e Pratiche. L'applicazione dei criteri di maturità TMMi migliorerà il processo di test e ha dimostrato di avere un impatto positivo sulla qualità del prodotto, sulla produttività dell'ingegneria dei test e sull'effort relativo del cycle-time. Il TMMi è stato sviluppato per supportare le organizzazioni nella valutazione e nel miglioramento dei loro processi di test.

TMMi dispone di un'architettura a fasi per il miglioramento dei processi. Contiene fasi o livelli attraverso i quali un'organizzazione procede mentre il proprio processo di test evolve da uno ad hoc e non gestito a uno gestito, definito, misurato e ottimizzato. Il raggiungimento di ogni fase garantisce che tutti gli obiettivi di quella fase siano stati soddisfatti, ponendo i miglioramenti come base per la fase successiva. La struttura interna del TMMi è ricca di pratiche di testing che possono essere apprese e applicate in modo sistematico per supportare un processo di test di alta qualità che migliora gradualmente.

Ci sono cinque Livelli nel TMMi che prescrivono la gerarchia di maturità e il percorso evolutivo per il miglioramento del processo di test. Ogni livello comprende una serie di Aree di Processo che un'organizzazione

deve implementare per raggiungere la maturità a quel Livello. Le Aree di Processo per ogni Livello di Maturità TMMi sono illustrate nella Figura 1.

Uno dei principi fondamentali alla base del TMMi è che si tratta di un modello generico, applicabile a diversi modelli del ciclo di vita e ambienti. La maggior parte degli Obiettivi e delle Pratiche definiti nel TMMi si è dimostrata applicabile sia nei modelli del ciclo di vita sequenziali che in quelli iterativi, incluso Agile. Tuttavia, al livello più basso del modello, molte Pratiche Secondarie ed esempi forniti risultano (molto) differenti a seconda del modello del ciclo di vita adottato.

Va sottolineato che, all'interno del TMMi, solo gli Obiettivi sono obbligatori, mentre le Pratiche non lo sono. Il TMMi è disponibile gratuitamente sul sito della TMMi Foundation ed è anche disponibile in formato libro pubblicato. Il modello è stato tradotto in molte lingue, tra cui spagnolo, francese e cinese.

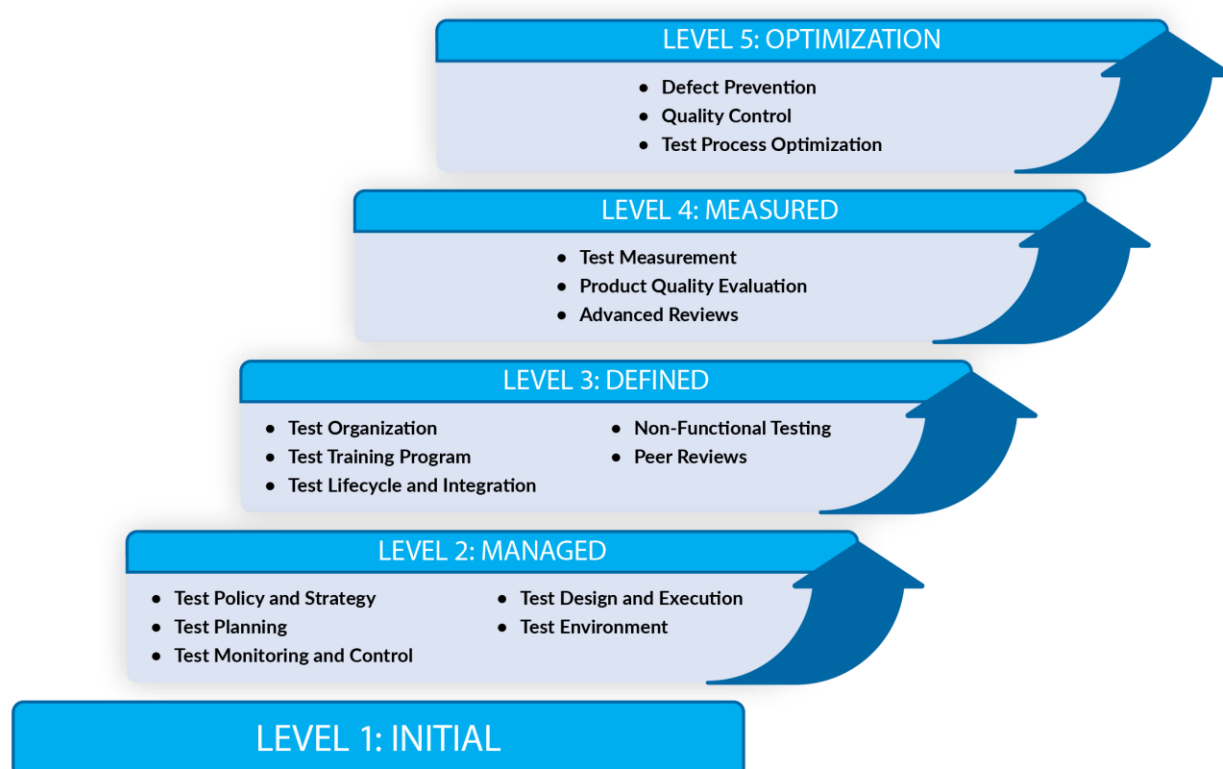


Figura 1: Livelli di Maturità TMMi e relative Aree di Processo

1.4 Testing in un contesto DevOps

1.4.1 Le sfide del testing in un contesto DevOps

Il testing in un contesto DevOps presenta sfide specifiche, dovute alla natura delle pipeline di Continuous Integration e Continuous Delivery (CI/CD), alla collaborazione tra i team di sviluppo e operations e al ritmo rapido dei cicli di sviluppo. Alcune delle principali sfide nel testing sono:

- **Continuous Testing:** i test devono essere eseguiti in modo rapido e frequente per tenere il passo con le pipeline di CI/CD. C'è una forte dipendenza dal testing automatizzato, che richiede test script e framework di test robusti e affidabili.

- **Gestione dell'Ambiente:** garantire una corretta gestione dell'ambiente di test e il provisioning di ambienti di test realistici per il continuous testing può risultare complesso. Spesso manca parità tra gli ambienti di produzione e di test, il che può portare a problemi non rilevati. Un'altra sfida è rappresentata dalla gestione e dal mantenimento di configurazioni consistenti tra ambienti diversi. Gli ambienti dovrebbero essere distribuiti e configurati automaticamente, di solito su richiesta, e rimossi dopo il testing.
- **Gestione dei Dati di Test:** mentre i dati di test dovrebbero essere simili a quelli di produzione, consistenti e distribuiti su richiesta, è anche essenziale rispettare le normative sulla privacy.
- **Testing di Integrazione e Regression Testing:** integrare più componenti e servizi che presentano interdipendenze complesse, rendendoli difficili da testare. È inoltre fondamentale garantire che le nuove modifiche non compromettano le funzionalità esistenti, il che richiede regression test suite complete.
- **Strumenti e Infrastruttura:** integrare vari strumenti di testing con le pipeline di CI/CD e con altri strumenti DevOps, assicurando al contempo che l'infrastruttura del testing sia in grado di scalare per gestire l'aumento del carico e della complessità.
- **Sfide Culturali e Organizzative:** promuovere la collaborazione tra sviluppo, testing e operations, che tradizionalmente possono aver operato in silos con incentivi spesso contrastanti. È fondamentale che i membri del team possiedano le competenze necessarie per gestire il testing automatizzato, la continuous integration e le pratiche di deployment.
- **Performance Testing:** simulare carichi e condizioni del mondo reale in un ambiente di test controllato, ma anche integrare il performance testing nelle pipeline di CI/CD per monitorare e ottimizzare continuamente le prestazioni.
- **Testing di Sicurezza:** nelle pipeline ad alta velocità, il testing di sicurezza viene spesso trascurato. Integrare il testing di sicurezza (DevSecOps) senza rallentare lo sviluppo e garantendo che le vulnerabilità vengano rilevate precocemente richiede strumenti e strategie specializzati, rappresentando una sfida per molti team. Il testing di sicurezza dovrebbe preferibilmente essere integrato precocemente e lungo tutto il ciclo di vita dello sviluppo, utilizzando strumenti automatizzati per eseguire test di sicurezza statici e dinamici.
- **Feedback e Reporting:** fornire al team un feedback rapido, chiaro e azionabile basato sui risultati dei test. Generare report completi che forniscano informazioni sulla qualità della build, mantenendo al contempo il processo Agile.
- **Pratiche di Shift Left:** migliorare la qualità spostando le attività a sinistra il più presto possibile nel ciclo di vita. Lo Shift Left testing implica eseguire il test più precocemente nel processo, ad esempio mediante review (del codice), analisi statiche e uno unit testing approfondito.
- **Pratiche di Shift Right:** poiché distribuire le modifiche in produzione il più frequentemente possibile è un elemento chiave, una strategia di test deve bilanciare le attività di testing nella parte di CI con l'osservazione dell'utilizzo del sistema in produzione nella parte di CD.
- **Monitoraggio e Feedback Loop:** in DevOps, il testing non si conclude con il deployment; il monitoraggio continuo e la raccolta di feedback dagli ambienti di produzione sono fondamentali. Configurare strumenti di monitoraggio efficaci e interpretare i dati provenienti dalla produzione per individuare difetti e ottimizzare le prestazioni è spesso un'attività trascurata o poco sviluppata.

1.4.2 TMMi e DevOps

È convinzione errata che TMMi e gli approcci DevOps siano in contrasto tra loro. In realtà, gli approcci DevOps e TMMi non solo possono coesistere, ma se integrati correttamente possono portare benefici significativi. Alcune delle sfide sono: cambiare il modo in cui si guarda al testing, essere completamente integrati nello sviluppo, e avere una pipeline di CI/CD e su cosa ciò significhi nel contesto di un programma di miglioramento dei “test”.

È importante notare che la “i” in TMMi si riferisce al fatto che il testing dovrebbe essere parte integrante dello sviluppo software, e non trattato come qualcosa di completamente separato. L’utilizzo del modello TMMi offre promemoria di pratiche di testing cruciali — ad esempio l’analisi del rischio di prodotto — che spesso vengono “dimenticate” in un contesto DevOps. Questo documento mostrerà, attraverso esempi concreti, come i metodi TMMi e DevOps possano lavorare insieme in modo efficace.

Quando si implementa il TMMi, è essenziale tenere presente che il modello non è pensato per “imporre” un insieme di pratiche a un’organizzazione, né deve essere applicato come uno standard al quale occorre “dimostrare conformità”. Se utilizzato correttamente, il TMMi può aiutare a individuare le aree specifiche del testing in cui un cambiamento può generare valore, in relazione agli obiettivi di business. Questo vale indipendentemente dal modello di ciclo di vita adottato.

È fondamentale ricordare sempre che le Pratiche TMMi sono un componente atteso, ma possono essere raggiunte anche attraverso ciò che viene definita una pratica “alternativa” rispetto a una pratica TMMi definita. Bisogna sempre chiedersi: qual è l’intento della pratica? Qual è la sua logica? In che modo aggiunge valore al business?

Spesso, in un ambiente DevOps, l’intento è già soddisfatto, ma mediante una pratica alternativa. In genere, “qualsiasi” soluzione è conforme finché è guidata da esigenze di business! Quando si adotta il TMMi, è bene evitare un approccio eccessivamente prescrittivo: non è questo lo spirito con cui il TMMi è stato concepito.

Interpretate sempre gli Obiettivi e le Pratiche TMMi nel contesto della vostra organizzazione. In generale, partendo dall’identificazione dei bisogni del processo all’interno del proprio contesto di business, poi si possono prendere decisioni su come focalizzare e guidare le priorità di miglioramento del processo.

La maggior parte dei conflitti che sorgono tra TMMi e DevOps si basano su una visione storica del TMMi di come dovrebbe apparire una “buona pratica” una volta implementata, oppure su un’incomprensione della logica delle pratiche DevOps e su come dovrebbero supportare i valori e il modo di lavorare tipico di DevOps.

Gli esperti TMMi, inclusi gli assessor e lead assessor, dovranno rivalutare e potenzialmente ripensare i messaggi che potrebbero essere condivisi inavvertitamente in relazione a come dovrebbe apparire una “buona pratica conforme al TMMi” quando viene implementata. Quando gli approcci DevOps vengono implementati correttamente in combinazione con i processi TMMi, ciò porta all’implementazione efficace delle pratiche di testing, non alla loro eliminazione.

Va inoltre notato che, oltre alle Pratiche Specifiche (del testing) all’interno del TMMi, esistono anche Pratiche Generiche. Lo scopo delle Pratiche Generiche è supportare l’istituzionalizzazione di un’Area di Processo, ovvero assicurare che l’organizzazione disponga di un’infrastruttura adeguata a sostenere quell’Area di Processo nel tempo, anche quando ci sono cambiamenti nel personale o altre trasformazioni organizzative.

La transizione dallo sviluppo software tradizionale al DevOps può rappresentare anche l'occasione per snellire e ridurre i processi così come sono definiti attualmente. In questo modo, le organizzazioni basate sul TMMi possono trarre vantaggio dal pensiero Agile e Lean, entrambi elementi presenti all'interno di DevOps.

Si è verificata una tendenza da parte delle persone a leggere nel modello TMMi elementi inesistenti, creando così processi e prodotti di lavoro non necessari e privi di valore aggiunto. Tornando alle origini e agli obiettivi di miglioramento, e utilizzando il modello TMMi nel modo in cui è stato concepito, si favorirà l'allineamento dei processi a reale valore aggiunto con le reali esigenze e obiettivi del processo.

Ridurre e snellire i processi con una mentalità Agile/Lean porterà a processi che riflettono ciò che le persone realmente fanno, e garantiranno anche che vengano raccolti solo i dati effettivamente utilizzati. Questa mentalità porterà inoltre a focalizzarsi sul mantenere le cose il più semplici possibile, il che in genere non è facile, ma porterà benefici a chi implementa il TMMi.

I miglioramenti all'interno di DevOps, guidati dalle metriche delle prestazioni DORA – *Change Lead Time (Lead time delle modifiche)*, *Deployment Frequency (Frequenza di deployment)*, *Change Fail Percentage (Tasso di failure delle modifiche)* e *Failed Deployment Recovery Time (Tempo di ripristino del deployment fallito)* [DORA16], [DORA24] – saranno generalmente attuati da team piccoli e competenti in grado di intervenire rapidamente, il che rappresenta un ulteriore modo in cui TMMi può beneficiare dall'applicazione di DevOps.

Sebbene TMMi sia completo, le organizzazioni non possono applicare ogni singolo elemento TMMi. Per avere successo, le organizzazioni devono identificare le proprie pratiche di testing più rilevanti e i miglioramenti che richiedono priorità e attenzione.

1.4.3 Relazione con il syllabus DevOps di ISTQB

TMMi e ISTQB hanno stretto un'alleanza per promuovere ulteriormente e in collaborazione la professione del Software Testing. Anche durante la stesura di questo documento, il comitato tecnico di TMMi e il gruppo di lavoro del syllabus ISTQB "Quality in DevOps" hanno collaborato attivamente.

Il TMMi è un modello di miglioramento dei test, ed in quanto tale fornisce un approccio di miglioramento predefinito, con priorità basate sulla struttura del modello stesso. Questo documento si concentra sul fornire un'interpretazione degli obiettivi di miglioramento di TMMi per chi opera in un contesto DevOps. Non fornisce una descrizione dettagliata completa delle pratiche ingegneristiche DevOps.

Il syllabus ISTQB "Quality in DevOps" è invece un documento basato sui contenuti, e come tale mira a fornire una descrizione approfondita delle buone pratiche ingegneristiche in ambito DevOps, come ad esempio le pratiche legate alla quality engineering (compreso il testing).

I due documenti sono quindi altamente complementari.

In sintesi:

- TMMi definisce che cosa dovrebbe essere migliorato (ovvero, quali processi)
- Il syllabus ISTQB DevOps descrive come le cose dovrebbero essere fatte (ovvero, le best practice ingegneristiche)

2 Livello 2 TMMi - Managed

2.1 Area di Processo 2.1 Test Policy and Strategy

Lo scopo dell'Area di Processo Test Policy and Strategy è sviluppare e stabilire una politica di test e una strategia di test a livello di organizzazione o di programma in cui le attività di test – come i tipi di test e i livelli di test – siano definiti in modo univoco.

Per misurare le prestazioni dei test, il valore delle attività di test e identificare le aree per il miglioramento, vengono introdotti dei performance indicator dei test.

2.1.1 SG1 Stabilire una politica di test

Ogni organizzazione che intraprende un progetto di miglioramento dei test dovrebbe iniziare definendo una politica di test. La politica di test definisce gli obiettivi generali dei test, i traguardi e le visioni strategiche dell'organizzazione in merito al testing e al ruolo dei professionisti del test.

È fondamentale che la politica di test sia allineata alle politiche di business e di qualità complessive dell'organizzazione. I miglioramenti dei test dovrebbero essere guidati da obiettivi di business chiari, i quali dovrebbero essere documentati nella politica (di miglioramento) di test.

Una politica di test è necessaria per ottenere una visione comune sul testing e sui suoi obiettivi da parte di tutti gli stakeholder all'interno di un'organizzazione. Questa visione comune è indispensabile per allineare le attività (di miglioramento del processo) di test in tutta l'organizzazione.

Si noti che gli obiettivi dei test non dovrebbero mai essere fini a sé stessi: derivano dall'obiettivo di livello superiore di stabilire software funzionante e qualità del prodotto.

Quanto detto sopra vale anche per un'organizzazione che ha abbracciato la cultura DevOps e adotta pratiche di sviluppo software DevOps.

Infatti, in molte organizzazioni esiste un ampio dibattito sul ruolo in evoluzione del testing, sull'indipendenza del testing, sulla test automation e sui tester professionisti nello sviluppo software DevOps. Questi e altri elementi sono problemi tipici che dovrebbero essere oggetto di confronto con il management e altri stakeholder, ed essere documentati nella politica di test.

Qualsiasi organizzazione, incluse quelle che operano con DevOps, che desidera avviare un progetto di miglioramento dei test deve prima identificare e definire gli elementi che guidano il business e le necessità per tale iniziativa. Altrimenti, perché iniziare un progetto di miglioramento?

Investire tempo per comprendere a fondo le reali esigenze di business consente di fornire un contesto adatto a decidere dove focalizzare le priorità di miglioramento (dei test), come ad esempio su quale Area di Processo.

Va notato che una politica di test è solitamente un documento snello di una sola pagina, una pagina web oppure un wall chart a livello organizzativo. Non è, quindi, un documento a livello di progetto.

L'Obiettivo Specifico di TMMi Stabilire una politica di test, con le relative Pratiche Specifiche, è pienamente applicabile anche alle organizzazioni che adottano lo sviluppo software DevOps.

Tuttavia, in DevOps, Sviluppo, Testing e Operations sono generalmente gestiti dallo stesso processo o, quantomeno, fortemente allineati. Di conseguenza, ci si aspetta che anche una politica di test sia parte integrante di una politica DevOps complessiva, focalizzandosi maggiormente sulla quality engineering piuttosto che solo sul testing.

In DevOps, il testing è organizzato bilanciando approcci Shift Left e Shift Right, utilizzando pratiche collaborative e una mentalità Test First. In DevOps, le attività di testing devono supportare tutte e tre le vie (come definito nel DevOps Handbook); il flow, il feedback, e il continuous experimentation and learning.

Come affermato, gli obiettivi dei test devono concentrarsi su un equilibrio chiaro tra Shift Left e Shift Right; proattivo, design e test first thinking, per costruire la qualità necessaria e le attività di testing prima del deployment (all'interno e attorno alla pipeline di CI/CD) per catturare i difetti; e misure reattive per ridurre i tempi di inattività del servizio in produzione, nel caso in cui sfugga un difetto; ad esempio un meccanismo di roll-back automatico per ridurre il downtime, il blue-green deployment, o il canary release per ridurre l'impatto.

Se da un lato l'obiettivo delle attività di testing è ridurre la probabilità di distribuire in produzione un difetto (software difettoso), dall'altro le strategie di deployment vengono applicate per ridurre l'impatto di eventuali failure, contribuendo così a una qualità complessiva migliore.

Per ottenere una riduzione dell'impatto, le attività di testing della pipeline di deployment devono essere semplificate, in modo da migliorare il flusso e ridurre i tempi di rilascio, mantenendo comunque bassi i livelli di rischio complessivo.

Oltre al testing durante lo sviluppo del software, in DevOps è pratica comune anche lo Shift Right (testing in produzione), e ciò influirà anche sulla definizione degli obiettivi di test come definiti nella politica di test, focalizzandosi su reazioni rapide ai problemi riscontrati in produzione, in base alle informazioni di telemetria, con l'obiettivo di ridurre il tempo di down-time, migliorare la disponibilità, e quindi innalzare la qualità.

Le pratiche di Site Reliability Engineering (SRE) coprono il monitoraggio dello stato di salute di sistemi, servizi e applicazioni, nonché la creazione di meccanismi di automazione closed loop per reagire a problemi di salute/anomalie. Questo approccio è noto anche come self-healing [Beyer].

A causa del numero elevato di cambiamenti previsti in DevOps, gli obiettivi di test si concentrano fortemente sul regression testing, sia per quanto riguarda gli aspetti funzionali, sia per quelli non funzionali.

Alcuni aspetti non funzionali, possono essere misurati in produzione (Shift Right) dopo il deployment, mentre altri – come ad esempio la sicurezza – devono essere testati prima di andare in produzione.

Esempi di obiettivi di test:

- Il testing è focalizzato sul garantire che le modifiche apportate forniscano il valore atteso (test di accettazione) e non compromettano nessuna funzionalità esistente (regression test).
- Il testing deve supportare gli obiettivi di sviluppo dell'organizzazione, come la riduzione del tasso di failure delle modifiche o l'aumento della frequenza di deployment.

Anche l'indipendenza e la definizione di processi di alto livello per ogni politica di test risultano tipicamente influenzate nel contesto DevOps.

Per quanto riguarda l'indipendenza, il testing è un ruolo all'interno dell'organizzazione di sviluppo, e viene spesso identificato con il termine DevTest, per indicare un doppio ruolo. Pertanto, l'indipendenza è solitamente bassa.

Le organizzazioni si affidano spesso a un cosiddetto team di piattaforma, composto da Software Development Engineers in Testing (SDET), il cui ruolo principale è quello di sviluppare e mantenere gli aspetti di testabilità del prodotto e le soluzioni di test automation, da mettere a disposizione dei ruoli DevTest.

Inoltre, potrebbero essere nominati alcuni team di test dedicati per aree più complesse, come il testing non funzionale o scenari complessi end-to-end. In questo caso, l'indipendenza è maggiore, ma la cooperazione rimane comunque stretta.

Per quanto riguarda la definizione di processi di test di alto livello, il testing è ormai parte integrante del processo di sviluppo o della strategia di CI/CD, e in quanto tale lo sarà anche il processo di test stesso.

È importante garantire che la politica di test, solitamente un documento snello e leggero, venga distribuita a tutta l'organizzazione / value stream / Agile release train / tribe. Poiché il testing è una parte cruciale del lavoro dei team Agile e DevOps, e non un'organizzazione separata, la politica dovrebbe essere conosciuta da tutte le persone coinvolte e interessate.

Nota:

¹ Il blue/green deployment è una strategia di deployment in cui vengono creati due ambienti separati ma identici. Un ambiente (blue) esegue la versione attuale dell'applicazione e un ambiente (green) esegue la nuova versione.

2.1.2 SG2 Stabilire una strategia di test

La strategia di test segue la politica di test e funge da punto di partenza per le attività di testing all'interno dei progetti. Una strategia di test viene tipicamente definita a livello di organizzazione o di programma. Una strategia di test tipica si basa su una valutazione del rischio di prodotto di alto livello e include una descrizione dei tipi di test e dei livelli di test da eseguire. Non è sufficiente, ad esempio, affermare semplicemente che in ogni progetto verranno eseguiti il testing di integrazione e il testing di accettazione. È necessario definire cosa si intende per unit testing e testing di accettazione, come vengono generalmente svolti e quali sono i loro obiettivi principali. L'esperienza dimostra che, quando si definisce e si segue una strategia di test, è probabile che si verifichi una minore sovrapposizione tra le varie attività di testing, con conseguente maggiore efficienza del processo di test. Inoltre, siccome gli obiettivi di test e l'approccio dei vari tipi di test e livelli di test sono allineati, è probabile che rimangano meno lacune, portando a un processo di test più efficace e quindi a un livello di qualità del prodotto più elevato.

Una strategia di test è un documento fondamentale in un ambiente DevOps. Essa definisce ad alto livello il testing che i team devono eseguire, principalmente all'interno della strategia di CI/CD; quali livelli di test e tipi di test vengono eseguiti e, ad alto livello, il loro approccio. Per ogni livello di test e tipo di test, vengono definiti gli obiettivi, le responsabilità, le principali attività di continuous testing e i criteri di accettazione/quality gate. La strategia di test segue solitamente i quadranti del testing Agile (vedi figura 2), ma è strutturata attorno alla pipeline di CI/CD per rappresentare il flusso delle attività all'interno della pipeline. La pipeline deve essere intesa non solo come le attività eseguite da uno strumento di CI/CD, ma come tutto ciò che i team fanno quando lavorano su un elemento del backlog. Attività di test come eseguire review e testing esplorativo avvengono al di fuori degli strumenti di CI/CD, ma come parte integrante del flusso/pipeline/value stream. Si

noti che la pipeline può essere utilizzata non solo come tecnologia/strumento, ma come rappresentazione visiva delle attività, del processo e delle persone (vedi figura 3).

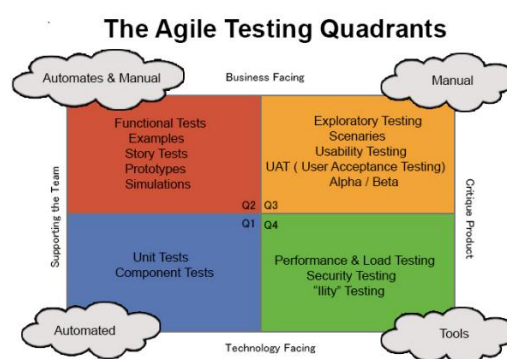


Figura 2: I quadranti del testing Agile

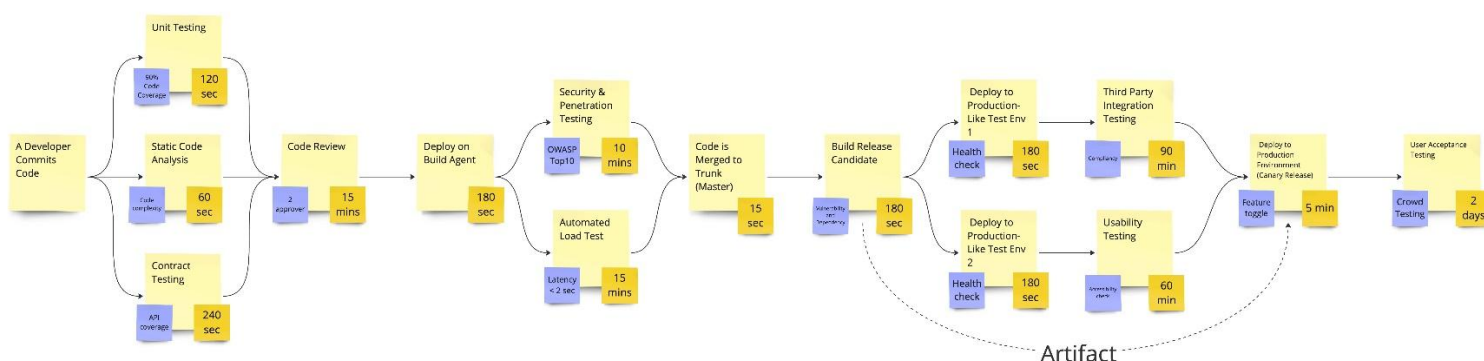


Figura 3: Una strategia di test rappresentata come una pipeline di alto livello

Una strategia di test è un documento fondamentale anche quando si adotta un approccio DevOps. Poiché DevOps è relativamente nuovo per molte organizzazioni, una strategia di test comune e a livello di organizzazione serve a guidare tutte le persone coinvolte, e a spiegare dove e come viene eseguito il testing nella pipeline. Inoltre, evita che ogni progetto o pipeline reinvesta risorse.

La strategia di test dovrebbe naturalmente essere pienamente allineata con la strategia di sviluppo, in quanto il testing è parte integrante della pipeline di sviluppo.

Alcune considerazioni (organizzate per argomento) da tenere presenti nello sviluppo di una strategia di test DevOps sono:

Livelli di test

- I livelli di test sono rappresentati principalmente come attività/lavori di test nella pipeline di CI/CD.
- La strategia di test viene spesso costruita attorno al testing di accettazione delle user story sviluppate, secondo pratiche di Behavior Driven Development (BDD). Questo richiede continuous testing in ogni fase della pipeline.
- Solitamente la strategia definisce un livello di test come il testing di contratti, per mitigare i rischi legati all'uso di API aperte e per supportare il testing di integrazione.

Testing di basso livello

- Le pratiche di sviluppo e di testing di basso livello, come pull request, merging e review del codice, devono essere prese in considerazione.

- Il testing statico e altre attività di test di basso livello sono solitamente imposte dallo strumento di CI/CD e spesso non possono essere evitate (ad esempio, controlli automatici e test come parte della pull request).
- In base al paradigma shift left, si pone particolare enfasi anche sullo unit testing, sul raggiungimento di un livello concordato di copertura del codice e sul testing di integrazione.

Test automation

- Il continuous testing è perfettamente integrato nella software delivery pipeline e nella toolchain DevOps.
- Una strategia di test automation è una parte cruciale della strategia di test DevOps.

Criteri di ingresso e di uscita

- I criteri di ingresso sono rappresentati dalla Definition-of-Ready (DoR) per i team DevOps. La DoR deve prevedere una base di test di alta qualità che serva come input per il testing, ad esempio i criteri di accettazione definiti in modo collaborativo dal team, solitamente nella forma di un'istruzione Given-When-Then. È importante che i rischi e gli obiettivi di test siano ben definiti e concordati prima di raggiungere lo stato Ready, affinché la stima copra anche le esigenze del testing.
- I criteri di uscita sono suddivisi tra la Definition-of-Done (DoD) per i team DevOps e il Quality Gate integrato in ogni lavoro di test nella pipeline di CI/CD. La DoD può essere definita in modo differente per i vari livelli di backlog (epic, feature, story), rappresentando il completamento del testing e il livello di confidenza raggiunto nei diversi livelli o fasi. Oltre al successo del testing di accettazione, la DoD include solitamente attività di testing previste per soddisfare i requisiti non funzionali, che costituiscono un vincolo per tutti gli elementi del backlog. Nei team DevOps, lo stato 'DONE' è raggiunto solo quando l'elemento del backlog è stato effettivamente distribuito in produzione e funziona correttamente in base al monitoraggio del sistema/servizio.

Tipi di test: Testing di sicurezza

- Identificare quali caratteristiche non funzionali, ad esempio, sicurezza, prestazioni, affidabilità o usabilità, sono critiche per il successo e il relativo approccio di test.
- Data l'importanza della sicurezza, lo Static Application Security Testing (SAST), il Dynamic Security Analysis Tooling (DAST), l'Interactive Application Security Testing (IAST), la Software Composition Analysis (SCA), il penetration testing e l'API security testing possono essere menzionati, come (parte dell')approccio del testing di sicurezza.

Ambienti di test

- Gli ambienti di test sono rappresentati per i vari stadi di testing nella pipeline di CI/CD (ad esempio, sviluppo, staging, preproduzione), ognuno dei quali rappresenta un focus differente. Sono di solito anche allineati ai diversi livelli di test identificati (es. unit, testing di integrazione, testing dei sistemi e testing di accettazione). Poiché DevOps si concentra su un'automazione pesante delle pratiche di deployment e configurazione per gli ambienti di produzione, adottando pratiche come il configuration-as-code, è naturale applicare le stesse pratiche nella progettazione e costruzione degli ambienti di test. Con l'uso di ambienti cloud-based, creare un ambiente di test on demand e cancellarlo dopo l'uso è una pratica comune, che porta a un ambiente di test standardizzato, controllato e stabile. Quando si utilizzano strategie di deployment come il Blue-Green deployment, l'ambiente di test dell'ultima versione del

prodotto può diventare l'ambiente di produzione dopo aver superato con successo i quality gate e la DoD. In questo caso, l'ambiente di produzione precedente viene riutilizzato come prossimo ambiente di test (scambiando lo scopo tra gli ambienti Blue e Green).

Il documento lean test strategy rappresenta spesso una buona soluzione per le organizzazioni DevOps e una valida alternativa a (project) test plan dettagliati. L'Obiettivo Specifico TMMi Stabilire una strategia di test, incluse le sue Pratiche Specifiche, è pienamente applicabile alle organizzazioni che applicano lo sviluppo software DevOps. Come per la politica di test, è importante assicurarsi che la strategia di test sia distribuita a tutta l'organizzazione / value stream / Agile release train / tribe.

2.1.3 SG3 Stabilire i performance indicator dei test

Gli obiettivi di business per il miglioramento dei test, come definiti nella politica di test, devono essere tradotti in un insieme di performance indicator chiave dei test. La politica di test e i relativi performance indicator forniscono una direzione chiara e un mezzo per comunicare i livelli attesi e raggiunti di prestazione dei test. I performance indicator devono indicare il valore del testing e del miglioramento del processo di test agli stakeholder.

Poiché gli investimenti nel miglioramento del processo richiedono il supporto del management nel lungo periodo, è fondamentale misurare quantitativamente i benefici di un programma di miglioramento, per mantenerne alta la motivazione. È importante sottolineare che questo Obiettivo Specifico TMMi riguarda la definizione di un numero limitato (ad esempio 2 o 3) di performance indicator dei test. Non riguarda la configurazione e l'implementazione un programma completo di misurazione, ma piuttosto la definizione di un insieme di indicatori di base che indichino come il valore del testing cambia nel tempo e nei diversi ambienti di consegna.

Così come in Agile, anche in DevOps l'attenzione è maggiormente rivolta al pensiero sistemico e team-based. Questo può portare a un conseguente ampliamento degli indicatori, includendo i performance indicator relativi ai processi di sviluppo e deployment, piuttosto che limitarsi esclusivamente agli aspetti specifici del testing stesso.

I principali performance indicator DevOps seguono tipicamente le metriche definite da DevOps Research and Assessment (DORA). Si noti che, al posto della copertura (di test), viene misurato il tasso di failure delle modifiche durante il deployment in produzione (percentuale di deployment falliti).

Esempi di metriche DORA:

- Metriche di software delivery performance: deployment frequency (frequenza di deployment), lead time for changes (lead time per le modifiche), time to restore service (tempo di ripristino del servizio) e change failure rate (tasso di failure delle modifiche)
- Operational Performance: metriche di affidabilità, ad esempio MTTR (Mean Time To Repair) e MTTF (Mean Time To Failure)

I performance indicator dovrebbero essere parte integrante delle attività di CI/CD, e i dati dovrebbero essere disponibili quasi in tempo reale, e accessibili a tutta l'organizzazione. Oltre alle metriche DORA, vengono utilizzate anche le metriche Flow per supportare il Value Stream Management/l'ottimizzazione. Le attività di testing possono essere misurate in linea con le metriche sopra citate, assicurandosi che il loro contributo sia comprensibile.

Esempi di metriche flow per il Value Stream Management in relazione al testing:

- Lead time di esecuzione dei test, tasso di successo o di failure delle attività di test all'interno della CI/CD per ogni modifica al codice
- Metriche Flow, come il lead time di esecuzione dei test contribuisce al lead time complessivo

La sfida consiste nel definire il giusto mix di indicatori legati all'approccio DevOps e al pensiero sistemico, fornendo al contempo una buona indicazione dei risultati delle prestazioni di un programma di miglioramento dei test basato su TMMi. L'Obiettivo Specifico TMMi Stabilire i performance indicator dei test, incluse le sue Pratiche Specifiche, è pienamente applicabile, sebbene i performance indicator selezionati e applicati possano avere un ambito e un raggio d'azione più ampi rispetto al solo testing.

Naturalmente, questo rende più complessa l'analisi e l'interpretazione dei performance indicator. In effetti, i performance indicator utilizzati potrebbero non essere chiamati performance indicator dei test. Questo è comunque accettabile nel contesto TMMi, purché contengano elementi legati al testing e siano usati per valutare i progressi compiuti effettuando il miglioramento dei test.

2.2 Area di Processo 2.2 Test Planning

Lo scopo di Test Planning è definire un approccio di test basato sui rischi identificati e sulla strategia di test definita, e stabilire e mantenere piani ben fondati per l'esecuzione e la gestione delle attività di testing.

Attenzione: la chiave per una pianificazione dei test efficace sta nel pensiero anticipato ("l'attività"), non nella definizione del test plan associato ("il documento").

Nei cicli di vita DevOps, si verificano tipicamente due tipi di pianificazione: la pianificazione della release e la pianificazione dell'iterazione. L'Area di Processo Test Planning del Livello 2 TMMi si concentra sulle attività legate al testing sia nella pianificazione della release che dell'iterazione. La pianificazione della release guarda alla release di un prodotto all'inizio di un progetto. Essa richiede un product backlog definito e può prevedere l'affinamento di user story più grandi in una raccolta di user story più piccole. Tuttavia, è importante ricordare che in DevOps e Agile si lavora generalmente con tempistiche fisse e contenuti flessibili. È fondamentale assicurarsi che vengano rilasciati solo gli elementi del backlog che sono stati testati in linea con l'approccio di test e la Definition-of-Done (DoD). La pianificazione della release fornisce la base per un approccio di test e per un test plan che copre tutte le iterazioni. I release plan sono ad alto livello, focalizzati sulla strategia o sull'approccio complessivi, piuttosto che su compiti dettagliati. Dopo la pianificazione della release, ha inizio la pianificazione dell'iterazione per la prima iterazione. La pianificazione dell'iterazione guarda alla conclusione di una singola iterazione ed è incentrata sul backlog dell'iterazione.

2.2.1 SG1 Eseguire un assessment del rischio di prodotto

Il testing esaustivo è impossibile, ed è sempre necessario fare delle scelte e stabilire delle priorità. Questo Obiettivo TMMi è quindi applicabile anche ai progetti DevOps. Nei progetti DevOps, deve essere effettuato un assessment del rischio di prodotto di alto livello, basato su un documento di vision del prodotto o su un insieme di obiettivi di prodotto di alto livello, durante la pianificazione della release.

Per ogni iterazione, deve essere eseguita una sessione sul rischio di prodotto più dettagliata, basato sulle user story o su altri requisiti per quell'iterazione, come parte della sessione di pianificazione dell'iterazione. Talvolta, questa attività viene già svolta durante le sessioni di affinamento, dove le user story vengono discusse e anche analizzate per individuare i rischi.

Il processo di assessment del rischio di prodotto in un progetto DevOps avrà un formato molto più lightweight rispetto a quello applicato in un progetto tradizionale che segue un modello di ciclo di vita sequenziale. Esempi

di tecniche lightweight di assessment del rischio di prodotto da utilizzare sono il risk poker [Veenendaal12] e ROAM (Resolved, Owned, Accepted, Mitigated) [Baah].

Durante la pianificazione della release o la pianificazione del program increment (PI planning), i rappresentanti del business che conoscono le funzionalità nella release forniscono una panoramica di alto livello delle funzionalità da sviluppare, con i requisiti non funzionali come vincoli. Tutto il team, incluso il/i tester, partecipa all'identificazione e all'assessment dei rischi.

Durante la pianificazione dell'iterazione, il team DevOps identifica e analizza i rischi di prodotto sulla base delle user story da implementare nell'iterazione successiva. Preferibilmente tutti i membri del team Agile e, possibilmente, anche altri stakeholder partecipano alla sessione sul rischio di prodotto. Il risultato è una lista prioritizzata di elementi del rischio di prodotto, che identifica le aree critiche per il testing. Questo, a sua volta, aiuta a determinare la quantità appropriata di effort di test da allocare per coprire ogni rischio con un numero sufficiente di test, e a sequenziare questi test in modo da ottimizzare efficacia ed efficienza del lavoro di testing da svolgere. Le attività stimate possono essere prioritizzate in base al livello di rischio di prodotto associato ad esse. Le attività associate a rischi più elevati dovrebbero iniziare prima e richiedere un effort di testing maggiore. Le attività associate a rischi più bassi dovrebbero iniziare più tardi e richiedere meno effort di testing.

Data l'importanza della sicurezza in molte organizzazioni, la modellazione delle minacce può essere utilizzata, in aggiunta a quanto sopra, per identificare i requisiti di sicurezza, per individuare le minacce alla sicurezza e le potenziali vulnerabilità, per quantificare la criticità delle minacce e delle vulnerabilità e per prioritizzare dei metodi di rimedio per mitigarle. Le minacce alla sicurezza vengono prioritizzate, utilizzando le linee guida per quantificare la probabilità e l'impatto di ogni minaccia per arrivare alla severità.

2.2.2 SG1 Stabilire un approccio di test

Un approccio di test viene definito per mitigare i rischi di prodotto identificati e prioritizzati. Per una specifica iterazione, gli elementi e le funzionalità da testare vengono identificati durante la pianificazione dell'iterazione. Questa attività si basa anche sul risultato della sessione sul rischio di prodotto.

La lista prioritizzata degli elementi da testare si riferisce tipicamente alle user story da testare nell'iterazione. Le funzionalità fanno riferimento, tra l'altro, alle diverse caratteristiche di qualità del software da testare. Nuovi rischi di prodotto possono emergere durante l'iterazione, richiedendo testing aggiuntivo. Questioni come i nuovi rischi di prodotto che richiedono testing aggiuntivo sono tipicamente discusse negli standup meeting giornalieri.

Con DevOps, l'approccio di test è definito sulla base del Continuous Testing attraverso la pipeline di CI/CD. Continuous Testing significa testare presto e frequentemente utilizzando l'Automation con feedback fruibile e adeguato per ciascuna fase della pipeline di delivery. Gli strumenti di Continuous Integration (CI) e Continuous Deployment (CD) vengono in genere richiamati automaticamente al momento del commit del codice e orchestrano le attività di costruzione, integrazione, testing, sicurezza, conformità e deployment. Le attività di testing all'interno della pipeline di CI/CD si basano solitamente sui livelli di testing identificati tramite i Quadranti del Testing Agile, e sono continuamente ottimizzate dai team.

L'approccio di test definito a livello di iterazione o di elemento del backlog per mitigare i rischi può coprire, ad esempio, l'esecuzione di review aggiuntive delle user story e dei criteri di accettazione, un effort del testing proporzionale al livello di rischio, la selezione delle tecniche di test appropriate in base al livello e al tipo di rischio. Può anche suggerire attività di testing da eseguire solo dopo il deployment, ad esempio il crowd testing in produzione per raccogliere feedback sulla user experience, oppure il testing delle prestazioni con traffico di utenti reali tramite dark launch. Nel contesto DevOps, il Behavior-Driven Development (BDD) e l'Acceptance

Test-Driven Development (ATDD) sono tecniche popolari per guidare l'identificazione e la progettazione dei test case. L'approccio di test a livello di release sarà a un livello molto più alto e dovrà basarsi su una strategia di test definita a livello di programma o di organizzazione. Spesso un approccio di test è mantenuto o visualizzato sulla wiki del team/progetto.

Un rischio importante sempre presente nello sviluppo iterativo è il rischio di regressione. L'approccio di test deve definire come viene gestito tale rischio. Tipicamente, questo viene fatto creando un set di regression test specifico, preferibilmente automatizzato. In questo contesto, è utile la piramide di test automation [Cohn]. Essa mostra come massimizzare il valore dell'automazione del regression test, partendo dagli unit test alla base della piramide e salendo verso il testing a livello di service. Lo user interface testing si trova all'estremità superiore. Gli unit test sono veloci e affidabili. Il livello service consente di testare la logica di business a livello di API o di service, senza l'onere dell'interfaccia utente (UI). Più alto è il livello, più il testing diventa lento e fragile. Poiché l'automazione è una componente importante di DevOps, la test automation è uno degli elementi principali dell'approccio di test DevOps, e non solo a supporto del regression testing. Un altro modo per affrontare il rischio di regressione è fare affidamento sull'osservabilità del sistema, monitorando continuamente il suo stato e il suo utilizzo, e costruendo meccanismi automatici per evitare rischi, ad esempio mediante roll-back a una versione precedente e stabile.

I criteri di ingresso (noti anche come Definition-of-Ready), normalmente parte di un approccio di test definito (Pratica Specifica 2.3), tendono a essere gestiti in modo diverso nello sviluppo DevOps. Nello sviluppo software DevOps, il testing è parte integrante del processo del team ed è un'attività continua. La checklist di criteri di ingresso precedentemente utilizzata viene suddivisa tra la Definition-of-Ready – ad esempio, avere i criteri di accettazione definiti come scenari di test utilizzando le istruzioni Given-When-Then – e i controlli all'interno della macchina di CI/CD, come ad esempio, i controlli effettuati in una pull request, l'esecuzione di uno smoke test su un ambiente di test appena distribuito prima di iniziare un'attività di esecuzione dei test più ampia, oppure la verifica che anche i dati di test necessari siano stati distribuiti. Di conseguenza, si dovrebbe implementare nel flusso di lavoro di CI/CD una checklist o un gateway specifico per determinare se il testing può o non può essere avviato, e dichiarare la build fallita in caso di non conformità, proprio come avviene per qualsiasi altro difetto. Questo vale anche per un componente che, all'interno di un team DevOps, passa da uno stage di test (ad esempio, unit testing) a un altro (ad esempio, testing di accettazione).

I criteri di uscita dei test (Pratica Specifica 2.4) fanno parte della cosiddetta Definition-of-Done (DoD), che si basa fortemente sui quality gate raggiunti all'interno della pipeline di CI/CD. È importante che la DoD contenga criteri specifici relativi ai test, ad esempio per la copertura dei test e la qualità del prodotto. L'iterazione dovrebbe portare all'implementazione del set concordato di user story e soddisfare i criteri di uscita (dei test) così come sono definiti nella DoD. In DevOps, "DONE" significa di solito che le modifiche al codice relative agli elementi del backlog sono state distribuite in produzione, superando tutti i quality gate all'interno della pipeline di CI/CD. Nel caso in cui la Release avvenga On-Demand, e sia separata dal Deployment tramite tecnologie come i Feature Toggle, esiste anche una DoD a livello di release, che può coprire più iterazioni. Tuttavia, l'obiettivo di DevOps è rilasciare spesso, in parti il più piccole possibile. La DoD a livello di release include in genere criteri legati alla copertura e alla qualità del prodotto, focalizzandosi sulla validazione del fatto che il valore atteso sia stato effettivamente fornito agli utenti, sulla base di attività di testing in produzione, come il crowd testing o il canary testing.

La Pratica Specifica 2.5 Definire i criteri di sospensione e ripresa è probabilmente non rilevante nei cicli di vita DevOps. Poiché il testing è parte integrante del processo di sviluppo software DevOps, non sarà ovviamente trattato come attività separata e indipendente dalle altre attività di iterazione. Quando si verificano problemi di blocco che possono essere considerati minacce potenziali o reali all'avanzamento del testing, questi vengono discussi durante lo standup meeting giornaliero. In questo contesto, il team decide quali azioni, se

necessarie, intraprendere per risolvere i problemi. Pertanto, criteri formali di sospensione e ripresa non sono richiesti né definiti: le problematiche vengono gestite come parte della normale routine DevOps. La routine DevOps rappresenta quindi una pratica alternativa a questa Pratica Specifica.

2.2.3 SG3 Stabilire le stime di test

Per i team DevOps, le stime dettagliate per il testing vengono effettuate durante la pianificazione dell'iterazione. Vengono effettuate stime (di test) di alto livello durante la pianificazione della release e possibilmente anche durante le sessioni di affinamento del backlog. Tutte le stime sono ovviamente effettuate come stime del team che includono tutto l'effort richiesto per distribuire ogni user story. È importante assicurarsi che le attività di testing siano effettivamente considerate durante le sessioni di stima. Questo può essere fatto identificando le attività di testing come compiti separati e successivamente stimando ogni singolo compito di test, oppure stimando ogni user story in cui il testing da eseguire è reso esplicito e quindi considerato. Poiché in DevOps ci si aspetta l'automazione di quasi tutto, anche l'automazione di attività relative al testing deve essere stimata, inclusa la test automation, l'automazione del deployment degli ambienti di test, l'automazione del deployment dei dati di test, nonché i miglioramenti della pipeline in linea con l'approccio di test identificato per l'iterazione. Durante la stima, un tester, essendo parte del team DevOps, deve partecipare alle sessioni di stima. Planning Poker, T-shirt sizing, swimlane sizing (bucket sizing) e dot voting sono tutte tecniche di stima tipiche utilizzate nello sviluppo software DevOps.

Il lavoro da svolgere per l'iterazione successiva è tipicamente definito dalle user story. Le user story devono essere di dimensioni ridotte affinché possano essere stimate. "Estimable" (Stimabile) è uno dei criteri delle user story definiti da INVEST ed è applicabile alle user story che devono far parte di un'iterazione. Durante la pianificazione dell'iterazione, i team Agile identificano e definiscono tipicamente i compiti relativi ai test. I compiti di test saranno registrati su una task board insieme agli altri compiti di sviluppo. Durante la retrospettiva, i team dovrebbero riflettere sulla precisione delle loro stime (se identificata come un problema) e migliorarla basandosi sull'apprendimento derivante dalla dimensione dei singoli compiti svolti nell'iterazione.

Durante la pianificazione della release, le user story o gli epic saranno tipicamente definiti a un livello più alto e non ancora dettagliati in compiti specifici. Ciò renderà ovviamente le stime più difficili e meno accurate. Come affermato, i progetti DevOps non stabiliranno una work breakdown structure (WBS) come base per le stime, ma potrebbero, a livello di release, beneficiare di informazioni su cosa e come sviluppare l'elemento del backlog, sufficienti a soddisfare l'accuratezza di stima necessaria, ovvero sufficienti a rendere possibile la prioritizzazione dell'elemento del backlog. Poiché in DevOps l'obiettivo è ottimizzare il flusso di valore, la stima deve supportare la decisione su cosa costruire successivamente, e non creare una timeline di progetto completa.

Sebbene un team DevOps effettui le stime in modo relativamente informale, la motivazione alla base delle stime dovrebbe essere chiara (cioè, quali fattori vengono considerati). Le discussioni basate su una motivazione promuovono un livello più elevato di accuratezza nelle stime. Tipicamente, nei progetti DevOps, la stima si concentra sulla dimensione (utilizzando story point) o sull'effort (utilizzando giorni uomo ideali come unità di stima). Normalmente, i costi non vengono affrontati come parte delle sessioni di stima nei progetti DevOps. Si noti che i costi infrastrutturali sono tipicamente elevati nel contesto DevOps; le loro stime di costo vengono tipicamente stabilite in sessioni di stima specifiche, non nelle sessioni di stima del team. Sono necessari uno stretto monitoraggio e controllo dei costi infrastrutturali, ad esempio l'utilizzo del cloud

pubblico, e ciò può essere integrato nella pipeline di CI/CD, con un reporting automatico (ad esempio, ogni lavoro di GitHub Action eseguito è presentato in minuti vCPU utilizzati).

Questo Obiettivo Specifico TMMi e le relative Pratiche Specifiche sono pertanto pienamente applicabili, ad eccezione della Pratica Specifica 3.2 Definire il ciclo di vita del test. Uno dei principi fondamentali dello sviluppo software iterativo e DevOps è lavorare in piccoli incrementi. Pertanto, le attività identificate sono in genere sufficientemente dettagliate da servire come base per la stima (dei test). Non vi è quindi necessità, nel contesto DevOps, di definire anche un ciclo di vita per le attività di testing come ulteriore base per la stima.

2.2.4 SG4 Sviluppare un test plan

Ribadiamo il concetto che la pianificazione dei test riguarda il pensiero anticipato ("l'attività") e non la definizione del test plan associato ("il documento"). In DevOps, la maggior parte delle attività di pianificazione dei test, come definite da questo Obiettivo Specifico TMMi, verranno svolte durante la pianificazione della release e dell'iterazione, nonché nelle sessioni di affinamento degli elementi del backlog. Tuttavia, il risultato di queste attività in genere non sarà documentato in un test plan, specialmente per la pianificazione dell'iterazione, dove potrebbero essere riflesse sulla task board. Poiché il testing viene eseguito come parte integrante della pianificazione della release e dell'iterazione, le "schedulazioni" risultanti includeranno anche le attività di testing. Piuttosto che una schedulazione dettagliata come quella sviluppata con cicli di vita sequenziali, la schedulazione all'interno di un progetto DevOps assomiglia molto di più a un ordinamento delle user story (elementi del backlog) e dei compiti che riflettono le priorità della release e dell'iterazione, ad esempio, in base alla consegna desiderata del valore di business. Va tenuto presente che l'esecuzione dei test avviene principalmente all'interno della pipeline di CI/CD, attivata dal commit del codice. Inoltre, il testing manuale o le attività svolte in produzione possono essere pianificati sulla board dell'iterazione. Le mappe mentali sono spesso utilizzate qui come tecnica di supporto. La task board rifletterà le priorità dell'iterazione. Pertanto, non viene stabilita nessuna schedulazione (di test) esplicita; ci si aspetta che siano definite chiaramente le priorità di release e di iterazione per le user story, rispettivamente per i compiti da eseguire, comprese le attività di testing.

A livello di progetto, il personale di test fa parte della costruzione del team; la necessità di risorse di test o multi-qualificate viene identificata anticipatamente. Man mano che i progetti cambiano o crescono, è facile dimenticare la motivazione iniziale per la selezione di un individuo in un determinato team o progetto, che spesso include esigenze di competenze specifiche o esperienze specificamente correlate al team o progetto. Ciò fornisce una valida ragione per documentare le competenze richieste alle persone, offrendo informazioni di supporto sul motivo per cui sono state selezionate per il team o progetto. Una volta definiti i team DevOps, il personale di test è più o meno fisso. Durante la pianificazione dell'iterazione, l'identificazione delle risorse e delle competenze (aggiuntive), ad esempio, per il testing non funzionale, necessarie per eseguire il testing in un'iterazione può, se necessario, essere discussa per garantire che il team disponga di risorse, conoscenze e competenze di testing sufficienti per eseguire i test richiesti.

Una sessione iniziale di rischio di progetto dovrebbe far parte della pianificazione della release e dell'iterazione. L'identificazione (e la gestione) di ulteriori rischi di progetto durante l'iterazione è parte degli stand-up meeting giornalieri e degli scrum-of-scrums, e viene tipicamente documentata tramite un registro degli impedimenti (impediment log). È importante che anche le problematiche legate al testing siano annotate nel registro degli impedimenti. Gli impedimenti

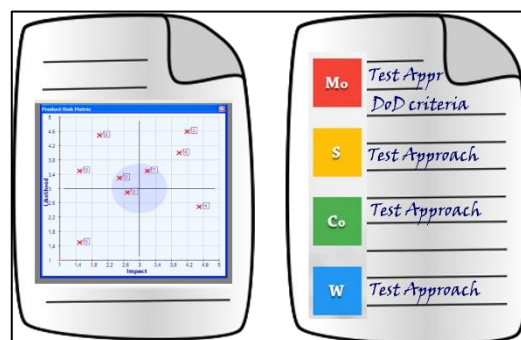


Figura 4: Un test plan su una pagina

dovrebbero essere discussi durante gli stand-up meeting giornalieri fino alla loro risoluzione, oppure negli scrum-of-scrums meeting, se il team non ha il livello o l'autorità necessari per risolverli.

Sebbene in genere non venga sviluppato e documentato un test plan specifico e dettagliato, la Pratica Specifica 4.5 Stabilire il test plan rimane comunque rilevante nel contesto DevOps. Tuttavia, i risultati delle discussioni che si svolgono nell'ambito della pianificazione dei test sono probabilmente catturati in una forma lightweight, possibilmente una mappa mentale o semplicemente un documento di una pagina (vedi Figura 4), un quadrante del testing Agile a livello di team e le attività di testing all'interno della pipeline di CI/CD.

2.2.5 SG5 Ottenere commitment per il test plan

All'interno di DevOps, il processo per sviluppare e stabilire un approccio di test e un test plan è un esercizio team-based, possibilmente guidato da un professionista del test (che fa parte del team stesso). La qualità del prodotto è una responsabilità del team. Pertanto, a condizione che il team segua il processo corretto e che le attività di testing concordate siano integrate nella pipeline di CI/CD, il commitment verso l'approccio di test e il test plan è già un risultato implicito della pianificazione della release e dell'iterazione, essendo un lavoro di squadra. Questo rappresenta ovviamente una differenza significativa rispetto al modo di lavorare in un ambiente tradizionale, dove tipicamente il tester prepara il test plan e successivamente deve ottenere un commitment esplicito. Le risorse dell'infrastruttura necessarie per la pipeline di CI/CD per implementare il test plan sotto forma di ambienti di test distribuiti e di test automatizzati eseguiti, devono essere considerate e fornite dagli stakeholder.

In un contesto DevOps, il team (incluso il Product Owner) deve comprendere e concordare l'elenco prioritizzato dei rischi di prodotto e le azioni di mitigazione dei test da eseguire. La comprensione e il commitment possono essere ottenuti, ad esempio, attraverso una breve presentazione seguita da una discussione durante la pianificazione della release o dell'iterazione, spiegando i rischi di prodotto, l'approccio di test e la sua logica al team.

Durante la sessione di stima (vedi SG 3 Stabilire le stime di test), il carico di lavoro viene stimato. Le risorse (di test) per un team sono fissate dalla configurazione del team DevOps. Le user story che vengono stimate e selezionate per essere sviluppate prendono le risorse disponibili come punto di partenza (vincolo). Pertanto, riconciliare i livelli di lavoro e risorse non è un'attività significativa; tuttavia, è fondamentale mantenere un focus continuo sulle risorse tecniche, come la disponibilità di calcolo/archiviazione/rete nel cloud, che devono essere fornite ai team. La Pratica Specifica 5.2 Conciliare lavoro e livelli di risorse è quindi una pratica tipicamente non rilevante in un contesto DevOps. Gli stand-up meeting giornalieri e gli Scrum-of-Scrums meeting saranno utilizzate per affrontare eventuali problemi di risorse durante l'iterazione e per (ri)allocare immediatamente le risorse appropriate o rimuovere un deliverable da un'iterazione per essere riconciliato nella pianificazione dell'iterazione o della release futura.

2.3 Area di Processo 2.3 Test Monitoring and Control

Lo scopo di Test Monitoring and Control è fornire una comprensione dell'avanzamento dei test e della qualità del prodotto in modo che possano essere intraprese azioni correttive appropriate quando l'avanzamento dei test si discosta in modo significativo dal piano e la qualità del prodotto si discosta in modo significativo dalle aspettative.

Ci sono alcuni aspetti che sono fondamentalmente diversi nel monitoraggio e controllo nel contesto DevOps rispetto ai progetti tradizionali. Sebbene il monitoraggio e il controllo siano elementi essenziali all'interno di DevOps, ciò non implica che aderire rigidamente a un piano sia l'obiettivo. Da una prospettiva di monitoraggio

e controllo dei test, ciò significa che non siamo guidati dal piano, ma revisioniamo costantemente i nostri progressi e risultati dal testing, e adattiamo il nostro piano e approccio come appropriato, ad esempio, man mano che diventano evidenti nuovi rischi di prodotto.

Il test plan, che solitamente è distribuito tra criteri di accettazione affinati, elementi del backlog (inclusi i loro livelli di rischio) e attività, è un'entità vivente e necessita di essere continuamente sottoposta a review e aggiornata man mano che nuove informazioni diventano disponibili o vengono forniti feedback. Gli obiettivi per le attività di testing sono definiti dall'insieme dei criteri di accettazione per il dato elemento del backlog, dall'approccio di test del team che viene aggiornato per il determinato elemento del backlog e, a livello di backlog, vengono definiti criteri specifici di Definition-of-Done. I progetti DevOps devono anche tenere presente il 'quadro generale' e monitorare e controllare sia a livello di CI/CD che a livello di iterazione.

È importante notare che, poiché il testing è un processo pienamente integrato nel processo complessivo del team DevOps, il monitoraggio e il controllo dei test sono anch'essi parte integrante dei meccanismi complessivi di monitoraggio e controllo del team DevOps. Di conseguenza, i tester non forniscono report a un test manager come nei progetti tradizionali, ma al team.

2.3.1 SG1 Monitorare l'avanzamento dei test rispetto al piano

I team DevOps utilizzano diversi metodi e strumenti per monitorare e registrare l'avanzamento dei test, ad esempio la progressione delle attività di test e delle story sulla Agile task board, e le burndown chart. Queste informazioni possono poi essere comunicate al resto del team tramite media come wiki dashboard, email in formato dashboard, oppure verbalmente durante gli stand-up meeting giornalieri. I team possono usare le burndown chart sia per tracciare l'avanzamento lungo l'intera release, sia all'interno di ciascuna iterazione. Una burndown chart mostra tipicamente l'avanzamento rispetto alla velocity attesa e al backlog delle funzionalità da implementare (e quindi mostra le prestazioni del team). In aggiunta, in DevOps si utilizza il monitoraggio continuo con dashboard online per monitorare in modo automatizzato infrastruttura, applicazione e rete lungo tutta la pipeline di CI/CD. Le risorse dell'ambiente di test vengono monitorate continuamente rispetto a quelle concordate durante la pianificazione dell'iterazione e per l'utilizzo nella pipeline di CI/CD. Un'altra pratica comune consiste nel tracciare i problemi relativi all'ambiente di test tramite la task board, ad esempio utilizzando adesivi con l'indicazione "ambiente di test bloccato" sulle story card, oppure creando una colonna separata sulla board dove vengono posizionate tutte le story bloccate dall'ambiente, in attesa che siano sbloccate. Tutto ciò serve a rendere visibile l'impatto di un ambiente di test che sta bloccando l'avanzamento e a indirizzare l'attenzione del team verso la risoluzione del problema, prima di lavorare su qualsiasi altra attività.

Per fornire una rappresentazione visiva immediata e dettagliata dello stato attuale del team e della CI/CD, incluso lo stato del testing, i team utilizzano in genere le task board. Le story card, le attività di sviluppo, le attività di test e le altre attività creati durante la pianificazione dell'iterazione sono riportati nella task board. Durante l'iterazione, l'avanzamento è gestito tramite lo spostamento di queste attività sulla in colonne, come "to do" (da fare), "work in progress" (in corso) e "done" (completate). Per mantenere il flusso, si applicano spesso dei limiti di Work in Process (WIP) ai vari stati di una Kanban Board, controllando il comportamento del team affinché completi le attività (in corso) invece di iniziare nuove attività. I team DevOps utilizzano solitamente strumenti per mantenere le proprie story card sulle task board, che possono automatizzare le dashboard e la visualizzazione dello stato. Tuttavia, alcuni team non creano attività specifiche per le attività singole, ma usano semplicemente la story card e annotano in essa dei commenti relativi ai test, facendo riferimento a strumenti o wiki dove il testing può essere documentato. Le attività di testing sulla task board sono generalmente collegate ai criteri di accettazione definiti per le user story e ai requisiti non funzionali che

costituiscono un vincolo per ogni elemento del backlog. Man mano che i test automation script e i test esplorativi relativi a un'attività di test raggiungono lo stato di superamento, l'attività viene spostata nella colonna "done" della task board.

La Definition-of-Done (DoD) funge da criterio di uscita in base al quale viene misurato l'avanzamento. La DoD dovrebbe essere anche collegata alle attività di testing e mostra tutti i criteri che devono essere soddisfatti prima che lo sviluppo e il testing di una user story possano essere considerati "Done". Si noti che i criteri del testing relativi alle attività di test costituiscono solo una parte di ciò che il team concorda di completare come Definition-of-Done. I criteri della Definition-of-Done vengono tipicamente applicati su più livelli, ad esempio a livello di story e a livello di epic. L'intero team revisiona regolarmente lo stato della task board, spesso durante gli stand-up meeting giornalieri, per assicurarsi che le attività si stiano muovendo attraverso la board ad un ritmo accettabile. Se alcune attività (incluse le attività del testing) non stanno avanzando o procedono troppo lentamente, ciò attiva una discussione team-based in cui vengono analizzati i problemi che potrebbero bloccare l'avanzamento di quelle attività.

Lo stand-up meeting giornaliero coinvolge tutti i membri del team DevOps, inclusi i tester. In questo meeting, i membri comunicano al resto del team il proprio stato attuale e l'avanzamento effettivo. Eventuali impedimenti che possono bloccare l'avanzamento dello sviluppo o dei test vengono comunicati durante gli stand-up meeting giornalieri, in modo che l'intero team ne sia a conoscenza e possa agire di conseguenza. In questo modo, il risk management di progetto è integrato in questi meeting quotidiani. A cadenza regolare, il team può decidere di eseguire un esercizio di tipo ROAMing per supportare il monitoraggio dello stato dei rischi di progetto [ROAM]. Qualsiasi rischio di progetto, inclusi quelli legati al testing, come ad esempio la mancanza di disponibilità degli ambienti di test, può essere comunicato e affrontato durante lo stand-up giornaliero. (Nota: il monitoraggio dei rischi di prodotto fa parte del monitoraggio della qualità del prodotto ed è quindi trattato successivamente nell'Obiettivo Specifico SG 2 Monitorare la qualità del prodotto rispetto a piano e aspettative.) Gli stand-up meeting giornalieri per il task management quotidiano, e le pratiche del team DevOps relative alla correzione del corso delle attività, sono buone tecniche comprovate che soddisfano l'intento delle Pratiche Specifiche TMMi nell'Area di Processo Test Monitoring and Control.

Alla fine dell'iterazione viene condotta una sprint review rispetto agli obiettivi di sprint concordati. Il team dimostra ciò che è stato raggiunto, ad esempio quali story o epic sono state completate. L'integrazione attraverso la pipeline di CI/CD è, naturalmente, continua. I risultati del testing, ad esempio rispetto alla Definition-of-Done, fanno parte della iteration review. Vengono organizzate demo con gli stakeholder per discutere il valore di business e la qualità del prodotto consegnato. Gli stakeholder sono rappresentati dal product owner durante la pianificazione dell'iterazione, le iteration review (demo) e le retrospettive. Il product owner è coinvolto attraverso discussioni sul product backlog e fornisce feedback e input per l'elaborazione delle user story e la progettazione dei test durante l'iterazione. Altri stakeholder sono coinvolti alla fine di ogni iterazione, durante l'iteration review. Non è richiesto un monitoraggio specifico del coinvolgimento degli stakeholder, poiché la loro rappresentanza da parte del product owner è integrata nel modo di lavorare Agile.

2.3.2 SG2 Monitorare la qualità del prodotto rispetto a piano e aspettative

Per il monitoraggio della qualità del prodotto vengono utilizzati in gran parte gli stessi meccanismi impiegati per il monitoraggio dell'avanzamento (vedi SG1 sopra). Il monitoraggio della qualità del prodotto tramite dashboard online non viene eseguito solo dai team durante lo sviluppo e il testing, ma è anche un processo continuo in cui la qualità dei prodotti software viene monitorata in produzione. I ruoli operativi monitorano

costantemente la qualità del prodotto utilizzando l'osservabilità, solitamente definita come Service Level Indicator (SLI). Regressioni o deviazioni nei SLI innescano azioni correttive per preservare la qualità e la user experience. In DevOps, per il monitoraggio dei rischi di prodotto, l'attenzione si concentra sulla revisione dell'elenco dei rischi di prodotto, inclusi i rischi/minacce alla sicurezza, durante meeting regolari, piuttosto che sulla revisione di una qualunque documentazione dettagliata dei rischi. I rischi di prodotto recentemente identificati o modificati, ad esempio come risultato di testing esplorativo, saranno discussi e le azioni di testing necessarie saranno concordate. Se necessario, può essere richiesto un esercizio aggiuntivo di ROAMing [ROAM]. Lo stato dei vari rischi di prodotto è tipicamente mostrato tramite chart sulla dashboard.

I team DevOps utilizzano metriche defect-based simili a quelle adottate nelle metodologie di sviluppo tradizionali, come i tassi di superamento/fallimento dei test, i tassi di scoperta dei difetti, i difetti trovati e corretti, per monitorare e migliorare la qualità del prodotto. Il numero di difetti trovati e risolti durante un'iterazione e quelli scoperti durante le attività di continuous integration, così come il numero di difetti non risolti che potrebbero diventare parte del backlog per l'iterazione successiva, dovrebbero essere monitorati durante gli stand-up meeting giornalieri. Per monitorare e migliorare la qualità complessiva del prodotto, molti team DevOps utilizzano anche sondaggi sulla soddisfazione del cliente per ricevere feedback sulla soddisfazione del prodotto rispetto alle aspettative del cliente.

I criteri di uscita del testing, ad esempio per la copertura dei test e la qualità del prodotto, fanno parte della Definition-of-Done (DoD). Il rispetto dei criteri di uscita concordati è tipicamente monitorato attraverso la task board, per cui una story può essere indicata come "done" solo se soddisfa i criteri della sua DoD, che solitamente includono il criterio che il dato elemento del backlog sia stato distribuito in produzione (è "done" quando è distribuito, e non viceversa). Tipicamente, vengono definiti criteri di uscita specifici per il continuous testing. I difetti sono monitorati continuamente come parte del continuous testing all'interno della pipeline di CI/CD. Il soddisfacimento delle caratteristiche di qualità attese del prodotto in produzione è monitorato utilizzando l'osservabilità. Quality gates automatizzati (basati su soglie definite) vengono spesso implementati in diverse fasi della pipeline DevOps per bloccare il deployment quando la qualità del prodotto non soddisfa le aspettative. Questi gate possono valutare metriche come il tasso di superamento dei test, i benchmark delle prestazioni, la copertura del codice e i risultati dell'analisi statica del codice.

Le prestazioni e l'affidabilità sono spesso importanti caratteristiche di qualità non funzionali e, in quanto tali, fanno parte del monitoraggio della qualità del prodotto. Il testing delle prestazioni (ad esempio, load e stress test) può essere integrato nella pipeline per monitorare le prestazioni del sistema rispetto ai benchmark attesi. Vengono attivati avvisi quando le soglie di prestazione vengono superate, notificando ai team di affrontare immediatamente i problemi. Inoltre, strumenti di monitoraggio (dei log) possono essere utilizzati per tracciare in tempo reale i tassi di errore, i memory leak e i crash di sistema in produzione. L'identificazione precoce di questi problemi può aiutare a garantire che il prodotto soddisfi le aspettative di affidabilità.

Lo stand-up meeting giornaliero è il meccanismo utilizzato per eseguire quasi continuamente review della qualità del prodotto. Vengono organizzate demo con gli stakeholder per eseguire la validazione e discutere il valore di business e la qualità del prodotto in fase di consegna. La qualità del prodotto viene verificata e validata rispetto ai criteri di qualità definiti nella DoD.

Seguendo l'Area di Processo Test Planning, le Pratiche Specifiche sul monitoraggio dei criteri di ingresso, sospensione e ripresa sono probabilmente non rilevanti. Per ulteriori informazioni, si faccia riferimento all'Obiettivo Specifico 2 Stabilire un approccio di test dell'Area di Processo Test Planning, dove è stata fornita una spiegazione del motivo per cui questi criteri sono in genere non rilevanti in un ambiente DevOps.

2.3.3 SG3 Gestire le azioni correttive fino alla chiusura

I team DevOps notano molto rapidamente problemi come deviazioni dalle aspettative in una burn-down chart e/o mancanza di avanzamento di attività (di test) e story sulla task board. Ancora più importante, se la qualità del software in produzione sta degradando, ciò viene rilevato immediatamente e solitamente attiva azioni correttive, come ad esempio un roll-back a una versione stabile nota. Questi e altri problemi, ad esempio quelli che bloccano l'avanzamento dei test, vengono comunicati durante gli stand-up meeting giornalieri, in modo che l'intero team ne sia consapevole. Questo attiva una discussione team-based in cui i problemi vengono analizzati. Il team, insieme al product owner, deciderà quali azioni correttive intraprendere. La gestione delle azioni correttive nei progetti basati su DevOps è principalmente responsabilità del team auto-organizzato. Il team può definire e implementare le azioni correttive appropriate, oppure segnalare eventuali problemi come "impedimenti" allo Scrum master. Le azioni correttive legate alla qualità del prodotto sono talvolta automatizzate, e le root cause vengono monitorate per evitarne la ricorrenza. Questo fa parte della cultura del miglioramento continuo del team DevOps. Lo Scrum master ha solitamente la responsabilità di supportare il team nella gestione dei problemi fino alla loro chiusura. Eventi tipici per discutere e gestire le azioni correttive sono gli stand-up meeting giornalieri e i meeting delle retrospettive. Le azioni correttive che sono state concordate possono essere gestite fino alla chiusura come "attività" o "elementi del backlog", tramite il product backlog o (all'interno dell'iterazione) tramite la task board.

2.4 Area di Processo 2.4 Test Design and Execution

Lo scopo di Test Design and Execution è migliorare la capacità del processo di test durante la progettazione e l'esecuzione dei test stabilendo le specifiche di progettazione dei test, utilizzando tecniche di progettazione dei test, eseguendo un processo strutturato di esecuzione dei test e gestendo gli incidenti di test fino alla chiusura.

Sebbene l'obiettivo di fondo ("mitigare i rischi e testare il software") sia lo stesso in un progetto DevOps e in un progetto tradizionale sequenziale, l'approccio su come eseguire i test è in genere molto diverso. In DevOps, flessibilità e capacità di rispondere al cambiamento sono punti di partenza fondamentali. Inoltre, l'analisi e la progettazione dei test, l'implementazione dei test e l'esecuzione dei test non sono fasi di test successive, ma vengono eseguite in parallelo, in modo sovrapposto e iterativo, supportate da cicli di feedback rapidi. Un'altra differenza chiave è il livello di dettaglio della documentazione dei test stabilita, che si basa principalmente sulla documentazione come parte del codice di automazione dei test case. Con il testing manuale, nei progetti DevOps vengono tipicamente utilizzate tecniche experience-based e defect-based, a complemento della test automation. Inoltre, il testing manuale può essere impiegato nel crowd testing, dopo il deployment del software in produzione e il rilascio a un gruppo preselezionato di persone, che seguono alcune istruzioni/test charter e forniscono feedback aggiuntivi allo sviluppo su specifiche caratteristiche di qualità, solitamente usabilità o accessibilità. Anche le tecniche di progettazione dei test specification-based possono comunque essere applicabili e utilizzate, in particolare come parte dell'analisi collaborativa dei requisiti, elencando possibili scenari di esempio per arricchire i criteri di accettazione delle User Story. Con un'enfasi maggiore su unit testing e testing di integrazione, le tecniche white-box come testing delle istruzioni e testing delle decisioni sono molto più diffuse, così come il testing di contratti sulle API. Un'ultima differenza rilevante è il livello di rischio di regressione derivante dall'elevata frequenza dei cambiamenti, che richiede un maggiore regression testing a vari livelli di test. Idealmente, il regression testing è altamente automatizzato, avviato ed eseguito automaticamente all'interno della pipeline di CI/CD, in più fasi di test. Ci sono molte differenze, ma alla fine, nel contesto dell'Area di Processo Test Design and Execution, si tratta sempre di creare test, mitigare i rischi di prodotto, eseguire i test e trovare i difetti.

2.4.1 SG1 Eseguire l'analisi e la progettazione dei test utilizzando tecniche di progettazione dei test

In DevOps, l'analisi e la progettazione dei test e l'esecuzione dei test sono attività che si supportano reciprocamente e che in genere vengono eseguite in modo continuo, integrate nel modo di lavorare Agile dell'intero team di prodotto. Nei progetti DevOps, i tester fanno parte di un team che crea e affina in modo collaborativo le user story. Vengono eseguite review informali frequenti durante lo sviluppo dei requisiti, inclusi i criteri di accettazione per ogni user story. Questi criteri vengono definiti in collaborazione tra rappresentanti di business, sviluppatori e tester. In genere, la prospettiva unica del tester contribuisce a migliorare la user story identificando dettagli mancanti, scenari alternativi e rendendola testabile. L'analisi dei test non è quindi un'attività separata esplicita, ma piuttosto un'attività implicita che i tester svolgono come parte del loro ruolo all'interno dello sviluppo collaborativo delle user story.

Sulla base dell'analisi delle user story, vengono identificate le condizioni di test. Dal punto di vista del testing, viene analizzata la base di test per capire cosa può essere testato – queste sono le condizioni di test [Veenendaal24]. Se i criteri di accettazione definiti sono sufficientemente dettagliati e chiari, possono anche sostituire il ruolo delle condizioni di test tradizionali. Spesso, nel contesto DevOps, tecniche come il Behavior Driven Development (BDD) e l'Acceptance Test Driven Development (ATDD), utilizzando la sintassi Gherkin, possono aiutare a definire scenari e identificare condizioni da testare, documentandole come criteri di accettazione della specifica user story. Allo stesso modo, per i requisiti non funzionali, le condizioni di test specifiche vengono spesso definite nel formato di criteri di accettazione. Le condizioni di test vengono poi tradotte in test, necessari per garantire la copertura dei criteri di accettazione definiti e concordati. Condizioni di test interessanti possono anche essere documentate nelle test charter da utilizzare con il testing esplorativo. Naturalmente, quanto sopra non si applica solo a livello di user story, ma anche al testing a livello di epic e feature. Le condizioni di test a livello di epic e feature sono solitamente più astratte rispetto a quelle a livello di user story e coprono più user story, oppure rappresentano requisiti non funzionali. Le tecniche di progettazione dei test specification-based sono generalmente utili per derivare condizioni di test a partire da user story e criteri di accettazione. Tuttavia, nel contesto DevOps, queste tecniche di progettazione dei test vengono spesso utilizzate in modo più implicito che esplicito, poiché i tester, grazie alla loro esperienza, le hanno interiorizzate e riescono a utilizzarle con flessibilità in base al contesto.

Con l'applicazione del principio test-first in DevOps, i test che coprono l'insieme delle condizioni di test vengono identificati (e possibilmente automatizzati) prima, o almeno in parallelo, allo sviluppo del codice. Questo approccio, a volte, può iniziare già durante l'affinamento del backlog. Per lo unit testing automatizzato, si può considerare un approccio come il Test Driven Development (TDD). Per livelli di test superiori, come già detto, il Behavior-Driven Development (BDD) e l'Acceptance Test-Driven Development (ATDD) sono approcci diffusi e fortemente collegati alla test automation.

Per la maggior parte del testing manuale, i test vengono identificati o affinati man mano che il team procede con l'esecuzione dei test. I test, nella maggior parte dei casi, non sono documentati con lo stesso livello di dettaglio dei progetti tradizionali, ma piuttosto sotto forma di idee di test, utilizzando il testing esplorativo. Per aree complesse e critiche, un approccio più tradizionale alla progettazione dei test e allo sviluppo dei test case (utilizzando tecniche formali di progettazione dei test) può essere il modo migliore per coprire i rischi. Tuttavia, anche con questo approccio, la quantità di documentazione sarà limitata rispetto a quella del testing in un ambiente a ciclo di vita sequenziale tradizionale. La prioritizzazione dei test segue la prioritizzazione delle user story che coprono. La prioritizzazione delle user story si basa sul valore di business: la priorità più alta è

correlata al valore di business più elevato. È importante che i test siano stabiliti per coprire sia i rischi funzionali sia quelli non funzionali, ma, specialmente in DevOps, è particolarmente importante coprire anche il rischio di regressione. La priorità dei test viene implementata aggiungendo test ai lavori della pipeline di CI/CD.

I dati di test specifici necessari a supportare le condizioni di test e l'esecuzione dei test vengono identificati. Tuttavia, in DevOps, a differenza degli ambienti tradizionali, i dati di test necessari in genere non vengono specificati in anticipo in un documento di specifica dei test. I dati di test vengono invece forniti non appena sono disponibili gli strumenti e/o le funzionalità necessarie. I dati di test vengono creati immediatamente per permettere l'avvio quasi immediato dell'esecuzione dei test manuali. Per i test automatizzati, tuttavia, i dati devono essere generalmente specificati in anticipo. I dati di test vengono di solito gestiti nello stesso modo degli ambienti di test e possono essere distribuiti a questi ambienti on demand. Gli strumenti di preparazione dei dati di test possono anonimizzare, randomizzare e mascherare i dati di produzione, secondo quanto richiesto dalle normative sulla privacy e dal GDPR.

La tracciabilità tra i requisiti, le condizioni di test e i test deve essere stabilita e mantenuta. I team devono rendere evidente di aver coperto, nell'ambito del testing, le diverse user story e i criteri di accettazione. Il team ha quindi bisogno del supporto di strumenti che consentano di organizzare e gestire i requisiti (user story) in modo da supportare l'assegnazione di identificatori a ciascuna user story, così che tali identificatori possano essere utilizzati per garantire che il testing sia completato secondo i criteri concordati. In genere, i test e le user story collegati tramite uno strumento offrono possibilità di misurazione in tempo reale della copertura. La tracciabilità tra test e story è visibile attraverso lo stato e il log del lavoro dei test nella pipeline di CI/CD, generati automaticamente.

2.4.2 SG2 Eseguire l'implementazione dei test

L'implementazione dei test consiste nel predisporre tutto ciò che è necessario per avviare l'esecuzione dei test. In genere, lo sviluppo della documentazione di test a supporto dell'esecuzione dei test è ridotto al minimo. Al suo posto, vengono sviluppati e prioritizzati (regression) test script automatizzati. La preparazione dei regression test inizia il prima possibile, in parallelo con le altre attività di testing.

In un ambiente DevOps, procedure di test dettagliate per il testing manuale non sono una pratica comune. Lavorando in un team competente, i test saranno con ogni probabilità documentati a un livello molto più alto di astrazione, ad esempio test charter con il testing esplorativo. Questo è sufficiente per chi esegue i test, poiché si presume che abbia il livello richiesto di conoscenza del dominio e competenza tecnica per eseguirli. Chi esegue i test lavora all'interno del team, quindi avrà una comprensione migliore sia di cosa è stato codificato, sia di come è stato fatto, e di come la funzionalità sia adatta allo scopo. Poiché il livello di cambiamento all'interno e tra le iterazioni è tipicamente elevato, sviluppare procedure di test dettagliate comporterebbe anche un notevole lavoro di manutenzione. In genere, gran parte del testing manuale viene eseguito utilizzando il testing esplorativo. Nel testing esplorativo, non vengono sviluppate procedure di test dettagliate, ma piuttosto test charter di alto livello da una pagina che descrivono le idee di test e le indicazioni per il tester. Tuttavia, la maggior parte dei team DevOps utilizza il testing automatizzato. Gli approcci tipici impiegati sono TDD, BDD e ATDD. Sebbene il BDD di per sé non sia per la test automation, può servire come base per l'automazione dei test di accettazione. Gli scenari scritti in BDD (con sintassi tipo Gherkin) sono tipicamente collegati a framework di test automation. Utilizzando uno o più di questi approcci, vengono creati i test script automatizzati, ad esempio per dettagliare gli scenari Gherkin, come parte dell'attività di implementazione dei test. I test script fungono anche da documentazione di test, grazie alla presenza di

commenti sufficienti per la comprensione e all'uso di buone pratiche di codifica. In genere, i test script sviluppati vengono raggruppati in test suite, che vengono successivamente collegate ai lavori di test nella pipeline di CI/CD.

I dati di test specifici, come definiti durante l'attività di analisi e progettazione dei test, vengono creati preferibilmente in formato ridistribuibile. I dati di test sono talvolta configurati come codice, ovvero questi dati vengono definiti sotto forma di file e regole dello schema. Un intake test viene specificato per essere implementato all'interno della pipeline di CI/CD. Questo test, talvolta chiamato confidence test o smoke test, viene utilizzato per decidere, all'inizio di ogni lavoro di esecuzione dei test nella pipeline, se l'oggetto di test è pronto per un testing più dettagliato e approfondito. Lo smoke test ha lo scopo di fornire un feedback immediato al team di sviluppo e agli stakeholder in caso di failure del test. Inoltre, per ogni ambiente o fase di test all'interno della pipeline di CI/CD, viene spesso definita una deployment health checklist automatizzata. Le attività relative alla preparazione dei dati di test, alle modifiche degli ambienti di test o all'implementazione della test automation vengono stimate e pianificate sulla task board del team durante la pianificazione dell'iterazione.

Per quanto riguarda la Pratica Specifica 2.4 Sviluppare la schedulazione dell'esecuzione dei test, l'esecuzione automatizzata dei test viene attivata da qualsiasi modifica al codice all'interno della pipeline di CI/CD. Gli scenari di attivazione vengono identificati e codificati (web-hook) per avviare un'esecuzione del test al momento della modifica. Le esecuzioni manuali dei test sono schedate in linea con la pianificazione dell'iterazione, gestite come attività di test tramite la task board e discusse durante gli stand-up meeting giornalieri.

2.4.3 SG3 Eseguire l'esecuzione dei test

I test vengono eseguiti secondo i lavori specificati nella pipeline di CI/CD a ogni modifica, e per le nuove story sviluppate dal team seguendo l'iteration plan. L'ordine dei lavori di test è definito per fornire un feedback rapido. Se sono disponibili risorse sufficienti, i lavori di test possono essere eseguiti in parallelo per amplificare il feedback. I costi e la sostenibilità del cloud sono aspetti chiave per ottimizzare la pipeline di CI/CD. I difetti/incident vengono segnalati sui report e vengono generati i test log. In un progetto DevOps, il software viene consegnato in produzione il più frequentemente possibile, attivato da modifiche al codice. Il testing è parte integrante del processo, nel quale il lavoro viene svolto in cooperazione per consegnare un prodotto di qualità. Gli smoke test vengono eseguiti come parte dei lavori di testing definiti nelle pipeline di CI/CD, prima dell'esecuzione di set di test più ampi. In caso di failure, l'esecuzione della pipeline non prosegue con il lavoro successivo, e viene fornito feedback allo sviluppo. Se la causa del failure viene identificata come un difetto, questo difetto viene segnalato in un report.

L'esecuzione dei test viene effettuata in linea con le priorità definite durante la pianificazione dell'iterazione. Alcuni test possono essere eseguiti utilizzando una test procedure documentata, ma generalmente molti test vengono eseguiti utilizzando il testing esplorativo e il testing session-based come framework. Le sessioni di test esplorativi si basano su test charter predefinite e sono time-boxed. I test possono anche essere eseguiti in produzione, prima che il software venga rilasciato a tutti gli utenti. Nel caso di crowd testing, un gruppo selezionato di beta tester fornisce feedback su alcuni aspetti qualitativi del prodotto prima della release su larga scala. Ciò richiede che le funzionalità possano essere attivate o disattivate (feature toggle) sulla base di prerequisiti predefiniti, come l'utente o la posizione geografica.

Con lo sviluppo DevOps, vi è una necessità crescente di organizzare e strutturare il regression testing. Il regression testing, in particolare a livello di unit test e di test di integrazione, fa parte di un processo di continuous integration. Si tratta essenzialmente di un processo automatizzato di “build and test” che avviene a ogni modifica del codice, rilevando rapidamente e precocemente i difetti. La continuous integration consente di eseguire regolarmente regression test automatizzati e di fornire feedback rapido al team sulla qualità del codice e sul livello raggiunto di copertura del codice. I test vengono inoltre eseguiti secondo i lavori specificati nella pipeline di CI/CD, attivati dalle modifiche al codice. Questi test script automatizzati forniscono i risultati dei test e i test log. I test log aggregati offrono in genere viste per epic, user story, test case, test suite, lavori di test e per l'intera pipeline.

Le discrepanze tra i risultati attesi e quelli effettivi vengono segnalate sui report come difetti. Il defect management è solitamente parte integrante del sistema di sviluppo e fornisce piena tracciabilità rispetto all'esecuzione e alle modifiche del codice. Inoltre, i test log vengono forniti automaticamente attraverso tutti i sistemi, per offrire un record cronologico dei dettagli rilevanti sull'esecuzione dei test.

Con lo sviluppo e il testing di nuove user story da parte del team, nei progetti Agile in genere si discute se tutti i difetti riscontrati debbano effettivamente essere registrati. In particolare, nel caso in cui il team sia co-locato, i tester devono parlare con gli sviluppatori, e i difetti che possono essere corretti immediatamente e inclusi nella build successiva potrebbero non dover essere registrati, ma semplicemente corretti. Alcuni team registrano solo i difetti che sfuggono alle iterazioni, altri li registrano solo se non possono essere corretti nella giornata stessa, altri ancora registrano solo i difetti ad alta priorità. Se non tutti i difetti riscontrati vengono registrati, devono essere disponibili dei criteri per determinare quali difetti dovrebbero essere registrati e quali no.

È considerata una buona pratica, anche nei team DevOps, registrare dati durante l'esecuzione dei test per determinare se l'elemento (o gli elementi) testati soddisfano i criteri di accettazione definiti e possono effettivamente essere etichettati come “done”. Il logging dei dati di test dovrebbe essere supportato dall'automazione in uso. Questo vale anche quando si applicano tecniche experience-based, come il testing esplorativo. Esempi di informazioni che potrebbero essere utili da documentare sono: copertura di test (quanto è stato coperto e quanto resta da testare), osservazioni durante il testing (ad esempio il sistema e la user story in fase di test sembrano essere stabili?), lista dei rischi (quali rischi sono stati coperti e quali restano tra i più importanti), difetti trovati e altri problemi, ed eventuali domande aperte. Le informazioni registrate dovrebbero essere raccolte e/o riassunte in una qualche forma di strumento di status management (ad esempio strumenti di test management, strumenti di task management, task board), in modo che il team e gli stakeholder possano comprendere facilmente lo stato attuale di tutto il testing eseguito. Ovviamente, al team non serve solo conoscere lo stato delle attività di testing in isolamento, ma piuttosto lo stato complessivo della user story. Un elemento del backlog è completo quando soddisfa la Definition-of-Done. Quando un team DevOps utilizza il Continuous Deployment come pratica, la Definition-of-Done include solitamente anche il completamento del deployment come criterio.

2.4.4 SG4 Gestire gli incident di test fino alla chiusura

Il termine “incident” in DevOps è tipicamente utilizzato per indicare un vero incidente in produzione che ha impatto sugli utenti. Pertanto, in questo paragrafo useremo il termine “difetto” per indicare un incident di test nel contesto del TMMi. Poiché i team DevOps sono anche responsabili di Operations, devono reagire rapidamente. L'incident management in DevOps è un processo di IT Service Management (ITIL): “La pratica di

minimizzare l'impatto negativo degli incident ripristinando il normale funzionamento del servizio nel più breve tempo possibile."

Poiché in DevOps quasi tutto è codificato, tutti i difetti rilevati sono in realtà difetti in qualche software. Se si tratta di un problema dell'ambiente, è un difetto nell'Infra as Code dell'ambiente; se si tratta di dati di test errati, è un difetto nel codice della test automation; se si tratta di altra inaffidabilità nell'esecuzione, può trattarsi di un difetto nel codice della pipeline, ecc. Come indicato nel paragrafo precedente, negli ambienti DevOps non tutti i difetti trovati verranno registrati. Questo Obiettivo Specifico si applica solo ai difetti che vengono registrati e che devono quindi essere gestiti fino alla chiusura. In linea di principio, la gestione degli incident in DevOps è semplice. Nel caso in cui l'incident venga registrato sulla task board, viene visualizzato come un difetto che blocca una story e le relative attività dal completamento. Il modo di lavorare DevOps standard verrà applicato come per qualsiasi altro impedimento o attività che sta bloccando l'avanzamento. Verrà discusso durante gli stand-up meeting e assegnato al team per la risoluzione.

Nel caso in cui si decida di posticipare un difetto rilevato a un'altra iterazione, esso diventa semplicemente un'altra opzione desiderata (o modifica) per il prodotto. Va aggiunto al backlog e prioritizzato di conseguenza. Quando la priorità è sufficientemente alta, verrà preso in carico dal team nell'iteration successiva.

È una buona pratica monitorare il numero di difetti trovati e risolti durante un'iterazione, così come il numero di difetti non risolti che potrebbero diventare parte del backlog per l'iterazione successiva, durante i (stand-up) meeting giornalieri e nei meeting di retrospettiva.

2.5 Area di Processo 2.5 Test Environment

Lo scopo di Test Environment è stabilire e mantenere un ambiente adeguato, inclusi i dati di test, in cui sia possibile eseguire i test in modo gestibile e ripetibile.

Con l'Area di Processo Test Environment viene stabilito e mantenuto un ambiente di test adeguato, inclusi i dati di test generici, in cui sia possibile eseguire i test in modo gestibile e ripetibile. Naturalmente, anche in un progetto di sviluppo software DevOps, un ambiente di test adeguato è indispensabile. Utilizzando le tecnologie cloud per ogni fase della pipeline di CI/CD, un ambiente può essere configurato on demand in uno stato controllato. A causa dei cicli brevi di un'iterazione, l'ambiente di test deve essere altamente stabile e disponibile. I problemi nell'ambiente di test, ad esempio nella pipeline di CI/CD, avranno sempre un impatto immediato sull'avanzamento e sui risultati dell'iterazione. Una corretta gestione della configurazione e delle modifiche all'ambiente di test e ai dati di test è pertanto di fondamentale importanza.

2.5.1 SG1 Sviluppare i requisiti dell'ambiente di test

La specifica dei requisiti dell'ambiente di test viene eseguita nelle fasi iniziali del progetto e successivamente in modo continuo, sulla base dell'apprendimento continuo. È importante comprendere il flusso del valore del prodotto, l'ambiente di produzione, il technical landscape e la tool chain di sviluppo utilizzati, per determinare le necessità dell'ambiente e sviluppare di conseguenza i requisiti che guidano l'implementazione dell'ambiente stesso. La specifica dei requisiti viene sottoposta a review per assicurarne la correttezza, l'adeguatezza, la fattibilità e la rappresentazione accurata di un ambiente operativo "reale". Una specifica anticipata dei requisiti ha il vantaggio di offrire più tempo per acquisire e/o sviluppare l'ambiente di test richiesto, in base a virtualizzazione, containerizzazione e altri aspetti cloud nativi frequentemente utilizzati nel contesto DevOps. Gli ambienti nella pipeline di build (la parte CI della pipeline) vengono di solito distribuiti su

scala ridotta e semplice, e sviluppati continuamente in linea con le crescenti aspettative di prestazioni e affidabilità. Tuttavia, nella pipeline di release (la parte CD della pipeline), potrebbe essere necessario un test end-to-end di un processo di business, e quindi di un ambiente di test corrispondente molto più complesso.

Le necessità degli ambienti di test vengono identificate e specificate durante le sessioni di affinamento, in linea con le pratiche e gli standard di infrastructure-as-code e configuration-as-code, come definiti nei playbook di CI/CD o da un team di piattaforma. La priorità dei requisiti relativi all'ambiente di test e ai dati di test è basata sulla priorità dell'elemento del backlog corrispondente. Talvolta viene specificato uno spike tecnico separato con criteri di accettazione, per definire requisiti ambientali più complessi. I requisiti sono tipicamente scritti in linguaggi conformi a Infrastructure/configuration-as-code, seguendo le pratiche e la terminologia di CI/CD dell'organizzazione. I requisiti sono mappati al value stream o agli stadi della pipeline di CI/CD. Le tool chain di sviluppo sono allineate alla tecnologia operativa e di deployment per garantire che siano necessarie e complete.

2.5.2 SG2 Eseguire l'implementazione dell'ambiente di test

L'implementazione della pipeline di deployment e degli ambienti di produzione e di test dovrebbe iniziare il prima possibile, in modo da avere una versione iniziale degli ambienti disponibile all'avvio della prima iterazione. Gli ambienti di test sono ambienti simili a quelli di produzione, che includono dati di test generici per eseguire i livelli di test rilevanti nelle determinate fasi di test di CI/CD. I team DevOps efficienti riutilizzano anche l'automazione della produzione per gli ambienti di test, distribuendoli on demand. Tutte le attività relative all'implementazione dell'ambiente dovrebbero essere gestite all'interno del backlog, come qualsiasi altro elemento del backlog.

Nel contesto DevOps, l'infrastructure-as-code è spesso utilizzata per definire e implementare i componenti dell'ambiente, e la configuration-as-code per definire e implementare le variabili e le configurazioni dell'ambiente, entrambi tramite script automatizzati e pipeline. È spesso importante avere la capacità per il provisioning on-demand degli ambienti, utilizzando funzionalità self-service all'interno delle pipeline di CI/CD, anche per le sessioni di test esplorativo. I dati di test generici per ambienti di codifica vengono creati utilizzando strumenti automatizzati per la generazione e la gestione dei dati di test. Uno smoke test viene generalmente eseguito all'interno della pipeline dopo il deployment di un'istanza dell'ambiente e prima che l'ambiente venga utilizzato per ulteriori attività.

2.5.3 SG3 Gestire e controllare gli ambienti di test

La gestione e il controllo degli ambienti di test e di produzione avvengono durante l'intero progetto DevOps. Gli ambienti sono gestiti e controllati per consentire un'esecuzione dei test senza interruzioni, utilizzando strumenti disponibili che permettono la creazione, la configurazione, l'allocazione e l'utilizzo degli ambienti e dei dati. Il management dei sistemi, eseguito sugli ambienti di test per supportare il provisioning, la configurazione, l'allocazione, l'utilizzo e lo smantellamento, comprende le seguenti pratiche:

- gestire l'infrastruttura tramite codice, automazione e monitoraggio
- gestire l'accesso all'ambiente di test fornendo le credenziali di accesso
- fornire supporto tecnico per problemi che ostacolano l'avanzamento durante l'esecuzione dei test

- fornire monitoraggio e telemetria, utilizzabili per analizzare i risultati dei test e le cause dei difetti

Anche il provisioning, l'allocazione, la correzione e l'uso dei dati di test sono gestiti e automatizzati ove possibile, a supporto del processo di test.

La creazione e l'allocazione degli ambienti di test sono automatizzate, coordinate e misurate per ottenere la massima disponibilità ed efficienza. Il provisioning degli ambienti in modalità self-service o on-demand (attivazione/disattivazione) e il monitoraggio delle istanze dell'ambiente, in particolare per gli ambienti persistenti, sono implementati. I problemi che si verificano durante il provisioning o l'utilizzo degli ambienti di test sono registrati e gestiti fino alla chiusura dai team DevOps. Gli incidenti relativi agli ambienti di test vengono registrati quando si osserva un problema, sia automaticamente come parte di test falliti, sia dagli utenti dell'ambiente.

Come conclusione principale per l'Area di Processo Test Environment, gli Obiettivi Specifici e le Pratiche restano tutti applicabili e non cambiano nella sostanza.

3 Livello 3 TMMi - Defined

3.1 Area di Processo 3.1 Test Organization

Lo scopo dell'area di processo Test Organization è identificare e organizzare un gruppo di persone altamente qualificate responsabili delle attività di test. Oltre allo svolgimento delle attività di test, il gruppo di test gestisce anche il miglioramento dei processi di test dell'organizzazione e degli asset di processo di test, sulla base di una comprensione approfondita dei punti di forza e di debolezza del processo di test e degli asset di processo attualmente in uso nell'organizzazione.

3.1.1 SG1 Istituire un'Organizzazione di Test

L'Organizzazione di Test è spesso un'area di processo fraintesa. Molti interpretano questo obiettivo come un requisito TMMi per l'esistenza di un gruppo di test indipendente, o addirittura di un reparto separato che svolga attività di test indipendenti. Poiché i team DevOps sono progettati per essere multifunzionali, con un focus sull'ottimizzazione del flusso e sull'amplificazione del feedback, un'organizzazione di test indipendente contrasterebbe questo principio. Di conseguenza, il gruppo di professionisti del test altamente qualificati menzionato in questo contesto è solitamente distribuito all'interno di team multifunzionali e su ruoli diversi. Tipicamente, un test competence centre o una test guild si concentra sul miglioramento dei processi di test a livello organizzativo e sulla crescita della professione del testing, incluse le competenze e i percorsi di carriera, mentre i team Scrum stabili nel tempo si focalizzano sull'ottimizzazione della generazione di valore. Entrambi rappresentano meccanismi di miglioramento.

Le attività e responsabilità tipiche di un cosiddetto test competence centre includono la definizione, la gestione e il miglioramento dei processi di test standard a livello organizzativo e l'assegnazione di membri con esperienza, conoscenze e competenze di testing ai team DevOps multifunzionali. Un test competence centre ha generalmente la responsabilità (ownership) delle metodologie e dei processi di test. Si noti che, nelle attività quotidiane, un tester fa tipicamente parte di un team DevOps ed è quindi coinvolto nel testing continuo all'interno della pipeline CI/CD; tuttavia, è comune che lo stesso tester appartenga anche a un'organizzazione di test, come un test competence centre o una test guild, focalizzata sulla professione del testing.

Un'organizzazione di test richiede una leadership sulle tematiche di testing e qualità. Tuttavia, il testing deve essere visto come parte integrante del flusso di valore e le attività di testing sono distribuite sui diversi ruoli dell'organizzazione DevOps. Per sviluppare le competenze di testing nei ruoli necessari, viene fornito supporto alla qualità da parte di uno specialista del test. L'opzione del test competence centre si integra bene con DevOps e garantisce che il testing venga preservato come disciplina e trattato con la dovuta serietà.

Un'altra opzione, in un contesto DevOps, è che l'organizzazione di test assuma la forma di una test guild. Una test guild è tipicamente un gruppo informale e auto-organizzato di tester, membri di diversi team DevOps. Attività importanti di una test guild possono includere la condivisione della conoscenza, l'apprendimento collettivo e il monitoraggio delle tendenze. L'adesione a una guild può essere volontaria. A una test guild possono essere assegnati anche compiti più formali, come la definizione dei percorsi di carriera nel testing, l'organizzazione della formazione e la definizione, pianificazione e implementazione dei miglioramenti dei processi di test. Un vincolo fondamentale per il successo di una test guild è che ai suoi membri venga assegnato tempo dedicato allo svolgimento delle attività della guild.

3.1.2 SG2 Istituire le Funzioni di Test per gli Specialisti del Test

Il testing è considerato una professione e una disciplina di valore. Le conoscenze e competenze di testing sono necessarie all'interno di un team DevOps. Lo specialista del test fornisce tali competenze e affianca (coach) gli altri membri del team durante lo svolgimento delle attività di test. Le conoscenze e competenze richieste a un tester DevOps tipico sono diverse rispetto a quelle richieste in un'organizzazione tradizionale. Il tester necessita ancora delle conoscenze e competenze di testing "tradizionali", ma deve inoltre possedere competenze nell'automazione dei test ed essere in grado di svolgere attività al di fuori del testing, ad esempio scripting o requirements engineering. In ambito DevOps, approcci come Behavior-Driven Development (BDD) e Acceptance Test-Driven Development (ATDD) sono ampiamente diffusi e devono quindi far parte del bagaglio di competenze del tester DevOps. Ulteriori competenze includono la progettazione e configurazione delle pipeline CI/CD, l'automazione del deployment, la gestione della configurazione e le strategie di deployment. Poiché il tester DevOps lavora in team, le soft skill sono importanti tanto quanto il continuo sviluppo delle competenze tecniche. Il tester DevOps è un cosiddetto tester a T, con più aree di conoscenza solide, che dovrebbero riflettersi nella descrizione delle funzioni di test. Alcuni tester possono essere più orientati alla tecnologia, mentre altri possono concentrarsi maggiormente sul design thinking e sulla comprensione del cliente. Alcune organizzazioni DevOps hanno definito tre diversi tipi di funzioni di test:

- **Software Development Engineer in Test (SDET)** – sviluppa soluzioni di automazione dei test e del deployment e funzionalità di testabilità del prodotto a supporto dei membri DevTest.
- **DevTest** – sviluppatori responsabili anche del testing del prodotto, in grado di analizzare i requisiti, progettare i test, implementarli ed eseguirli (di norma utilizzando gli strumenti di automazione e le funzionalità di testabilità fornite dagli SDET).
- **Tester (o Test Architect)** – supervisiona le attività di test tra i team DevOps, identifica e gestisce i rischi e, se necessario, esegue test specialistici sul prodotto, mantenendo una visione complessiva dello sforzo di test.

L'organizzazione di test, qualunque sia la sua forma, è composta da persone che possiedono le competenze e la motivazione per essere buoni tester in un contesto DevOps. Esse sono assegnate a specifiche funzioni di test. In particolare in ambito DevOps, poiché l'intero team svolge attività di test, è fondamentale definire anche per le funzioni non specificamente di test aspettative chiare in merito alle attività di testing, ai ruoli, alle responsabilità e alle conoscenze e competenze di testing richieste.

3.1.3 SG3 Definire i Percorsi di Carriera nel Test

I percorsi di carriera nel test vengono definiti per consentire ai professionisti del testing di migliorare conoscenze, competenze, status e riconoscimenti e quindi di progredire nella loro carriera, ad esempio passando da DevOps test engineer a DevOps test architect, test coach, oppure a ruoli come product owner, facendo leva sulle competenze di comprensione del cliente, o Scrum Master, basandosi sulle competenze interpersonali. Il percorso di carriera non deve necessariamente essere limitato al dominio del testing; ciò contribuisce, nel lungo periodo, a diffondere una mentalità orientata alla qualità in tutta l'organizzazione. Sulla base dei percorsi di carriera definiti, vengono sviluppati e mantenuti piani di sviluppo professionale individuali per ogni membro dell'organizzazione di test. Questo specifico obiettivo e le relative pratiche sono pienamente applicabili anche in un contesto DevOps. I team DevOps necessitano di tester che comprendano la complessità del proprio ruolo, sappiano affiancare gli altri membri del team e siano motivati; molti di questi aspetti sono affrontati proprio attraverso i percorsi di carriera nel test.

Tuttavia, la definizione concreta dei percorsi di carriera nel test in un'organizzazione DevOps differisce generalmente da quella adottata in un'organizzazione che segue un ciclo di vita sequenziale. Mentre nelle organizzazioni tradizionali i percorsi di carriera nel test sono spesso orientati a una crescita verticale (da tester a test manager), nelle organizzazioni DevOps tendono a concentrarsi su una crescita orizzontale (da junior tester a senior tester, a test architect e, eventualmente, a test coach) o su una crescita verso ruoli come Scrum Master o Product Owner.

3.1.4 SG4 Determinare, Pianificare e Implementare i Miglioramenti del Processo di Test

I miglioramenti del testing sono una componente fondamentale della mentalità di miglioramento continuo di un'organizzazione DevOps e del framework di Value Stream Management, basato su principi lean e system thinking. Le idee principali ruotano attorno all'amplificazione del flusso, all'eliminazione degli sprechi, all'amplificazione del feedback e all'utilizzo di dati e fatti raccolti dalla pipeline DevOps, ad esempio le Flow Metrics o le quattro metriche di performance DORA. Come per gli altri miglioramenti, anche i miglioramenti del testing vengono raccolti, prioritizzati e gestiti nel backlog dell'organizzazione.

Il processo di miglioramento descritto da questo specifico obiettivo TMMi e dalle relative pratiche è in gran parte coperto dalle riunioni di retrospettiva svolte dai team DevOps, che si basano sull'analisi di dati e fatti generati dalla pipeline. La retrospettiva rappresenta un'opportunità per il team DevOps di auto-valutarsi e definire azioni di miglioramento. Di norma, una retrospettiva si svolge dopo la review dell'iterazione e prima della pianificazione dell'iterazione successiva. Come parte della pipeline, vengono individuate metriche di progettazione e viene definita la raccolta dei dati, in modo che possano essere visualizzati sulla value stream map per comprendere dove si verificano ritardi, vincoli o problemi di qualità del prodotto. Queste metriche consentono di identificare opportunità di miglioramento (anche del testing). Mentre i team DevOps si concentrano su metriche più vicine alle loro attività, come i tempi di esecuzione dei test, i tassi di fallimento dei test di regressione o i tempi di deployment e configurazione degli ambienti di test, l'organizzazione nel suo complesso si concentra sulle Flow Metrics e sulle quattro metriche DORA.

Durante la retrospettiva, il team discute (anche sulla base delle metriche) cosa ha funzionato bene nell'iterazione, cosa può essere migliorato e quali azioni verranno intraprese nella successiva iterazione. Sono utili anche competenze di problem solving, ad esempio l'analisi delle cause radice mediante tecniche come il diagramma di Ishikawa o i "5 perché". Una cultura di apprendimento continuo incoraggia le persone ad acquisire costantemente nuove conoscenze e a scambiare esperienze e idee in ambito DevOps. Durante ogni retrospettiva, il team DevOps identifica e pianifica modalità per incrementare la qualità del prodotto, ad esempio migliorando i processi di lavoro o adattando i criteri della Definition of Ready o della Definition of Done a livello di team o di value stream. Le retrospettive possono portare a decisioni di miglioramento legate al testing, focalizzate su efficacia dei test, produttività del testing e qualità dei test. Possono inoltre riguardare la testabilità delle applicazioni, delle user story, delle funzionalità o delle interfacce di sistema. Tutti i membri del team, tester e non tester, possono fornire contributi sia sulle attività di testing sia su quelle non di testing. Al termine della retrospettiva, il team DevOps dovrebbe aver identificato i miglioramenti (di test) da implementare nell'iterazione successiva. L'implementazione di tali miglioramenti è responsabilità del team DevOps stesso e i miglioramenti vengono quindi gestiti come elementi di backlog. I tester assumono tipicamente la responsabilità dei miglioramenti del testing, assicurandone l'effettiva implementazione all'interno del team. La retrospettiva è un meccanismo fondamentale che consente al team di evolvere e migliorare continuamente durante l'intero ciclo di vita del progetto.

Poiché l'ambito è spesso limitato all'iterazione, vengono introdotti miglioramenti piccoli ma frequenti, principalmente focalizzati sulla risoluzione di problemi specifici del team. Tali miglioramenti non sono sempre orientati all'apprendimento trasversale tra value stream o all'istituzionalizzazione dei miglioramenti. Analizzando come il miglioramento dei processi (di test) è organizzato e gestito, si osserva spesso una minore enfasi su un gruppo di processo (di test) a livello organizzativo e una maggiore enfasi sull'autogestione del team. Sebbene questo non sia necessariamente negativo, è importante creare un meccanismo che consenta di condividere a livello organizzativo i miglioramenti locali di successo e di affrontare anche quelle aree di miglioramento che vanno oltre le possibilità del singolo team DevOps. Una possibile soluzione consiste nel prevedere la partecipazione di un rappresentante dell'organizzazione di test (test competence centre o test guild) alle retrospettive locali, per garantire che i miglioramenti vengano condivisi e istituzionalizzati quando opportuno.

Le organizzazioni di grandi dimensioni organizzano spesso retrospettive dedicate a livello di value stream, al fine di identificare potenziali problemi sistemici. Il responsabile del miglioramento dei processi di test può inoltre individuare problematiche più ampie o iniziative che impattano il testing a livello organizzativo e inserirle nel backlog del value stream.

3.1.5 SG5 Distribuire il Processo di Test Organizzativo e Incorporare le Lesson Learned

Anche nei progetti DevOps è importante poter accedere e riutilizzare, quando opportuno e con valore aggiunto, il processo di test standard dell'organizzazione e gli asset di processo di test. Tali elementi devono ovviamente essere allineati al modo di lavorare DevOps, con un focus sull'assurance della qualità e sul testing degli elementi di backlog a tutti i livelli. L'organizzazione di test garantisce che questi vengano distribuiti e adottati da tutti i team. (Si rimanda all'area di processo Test Lifecycle and Integration per ulteriori informazioni sul processo di test standard organizzativo e sugli asset di processo di test e sul loro utilizzo nei progetti DevOps). L'implementazione del processo di test standard dell'organizzazione e l'utilizzo degli asset di processo di test nei progetti DevOps possono essere monitorati dagli stessi team auto-organizzati, ad esempio affrontando il tema durante le retrospettive. Un approccio particolarmente efficace consiste nell'incorporare processi e template direttamente negli strumenti e nel rendere le pratiche obbligatorie tramite la pipeline, ad esempio con il rifiuto automatico delle pull request qualora non siano soddisfatti gli obiettivi di copertura o vengano violati gli standard di codifica concordati.

La presenza di un rappresentante dell'organizzazione di test, ad esempio un responsabile del miglioramento dei processi di test, in queste riunioni consente di monitorare lo stato di implementazione e di affrontare eventuali problematiche trasversali ai progetti. Questa figura può inoltre riportare all'organizzazione di test preziose lesson learned e miglioramenti del testing emersi dai team DevOps. Le lesson learned e i miglioramenti del testing possono successivamente essere formalizzati come proposte di miglioramento dei processi di test e gestiti di conseguenza.

3.2 Area di Processo 3.2 Test Training Program

Lo scopo dell'area di processo Test Training Program è sviluppare un programma formativo che favorisca lo sviluppo delle conoscenze e delle competenze delle persone, in modo che le attività e i ruoli di test possano essere svolti in modo efficace ed efficiente.

L'obiettivo principale del Programma di Formazione sul Test è fornire la formazione (di testing) necessaria ai tester e agli altri soggetti coinvolti nelle attività di test, sia all'interno sia all'esterno del team. Un programma di formazione efficace garantisce che coloro che partecipano alle attività di test continuino a migliorare le proprie conoscenze e competenze di testing e acquisiscano le necessarie conoscenze di dominio e altre competenze correlate al testing. Poiché la qualità e il testing sono una responsabilità del team nel suo insieme, ci si aspetta che la formazione sul testing non sia rivolta esclusivamente ai professionisti del test, ma all'intero team DevOps.

3.2.1 SG1 Istituire una Capacità Organizzativa di Formazione sul Test

Il processo inizia con l'identificazione dei fabbisogni strategici di formazione sul test dell'organizzazione. Tali fabbisogni ruotano attorno alla dotazione dei team delle competenze e delle conoscenze necessarie per implementare efficacemente le pratiche di testing all'interno delle pipeline di integrazione continua e distribuzione continua, garantendo al contempo che gli aspetti di sicurezza siano adeguatamente considerati. Oltre alle esigenze di formazione tradizionali e Agile per i tester, ad esempio sulle tecniche di progettazione dei test, sul testing esplorativo e sulle misure di copertura, emerge la necessità di acquisire conoscenze e competenze specifiche sul testing in ambito DevOps. Tra gli esempi rientrano strumenti e framework di automazione dei test, il security testing (ad esempio Static Application Security Testing – SAST, Dynamic Application Security Testing – DAST e penetration testing), Infrastructure-as-Code, pratiche di shift-left testing, Test Driven Development (TDD), Behavior Driven Development (BDD) e Acceptance Test Driven Development (ATDD). L'identificazione dei fabbisogni strategici di formazione sul test non è limitata ai soli tester; è infatti di fondamentale importanza definire anche i fabbisogni formativi legati al testing per gli altri membri dei team DevOps.

Come in qualsiasi altra organizzazione, i fabbisogni di formazione sul test vengono tradotti in un piano di formazione (snello), che può includere anche esigenze formative specifiche di progetto. Il piano di formazione deve essere riesaminato e devono essere assunti impegni in merito alla sua attuazione. Si noti che, in particolare in ambito DevOps, alcune competenze vengono trasferite in modo efficace ed efficiente attraverso modalità informali (ad esempio formazione on-the-job, sessioni di formazione durante la pausa pranzo, coaching e mentoring), mentre altre competenze richiedono una formazione formale. Inoltre, pratiche DevOps come Embed Knowledge e Continually Experiment influenzano l'approccio adottato per la progettazione e l'erogazione dei programmi di formazione.

3.2.2 SG2 Fornire la Formazione sul Test Necessaria

In linea generale, le pratiche specifiche dell'obiettivo SG2 "Fornire la formazione sul test necessaria" sono le stesse sia in un'organizzazione tradizionale sia in un'organizzazione DevOps. Sulla base del piano di formazione organizzativo, vengono individuati i membri del team che devono ricevere la formazione per poter svolgere efficacemente il proprio ruolo di test. La formazione viene quindi pianificata, incluse le risorse necessarie, ed erogata.

Come già indicato, in un'organizzazione DevOps vengono spesso utilizzate modalità di formazione informali. Ad esempio, oltre alla formazione formale, il mentoring e il coaching on-the-job sono tipicamente incoraggiati e adottati in modo continuativo. In effetti, il mentoring è considerato una responsabilità di tutti ed è atteso a tutti i livelli dell'organizzazione. Il lavoro in coppia è un'altra modalità comunemente utilizzata dai team per l'aggiornamento delle competenze e il trasferimento della conoscenza.

Vengono creati registri della formazione sul test svolta, sia informale sia formale, e viene valutata l'efficacia della formazione. Un corso di formazione (di test) seguito può essere valutato anche durante una

retrospettiva, nella quale si discute se le conoscenze e competenze dei partecipanti siano ora più adeguate allo svolgimento delle attività di test.

3.3 Area di Processo 3.3 Test Lifecycle and Integration

Lo scopo dell'area di processo Ciclo di Vita del Test e Integrazione è definire e mantenere un insieme utilizzabile di asset organizzativi di processo di test e di standard per l'ambiente di lavoro, nonché integrare e sincronizzare il ciclo di vita del test con il ciclo di vita dello sviluppo. Il ciclo di vita integrato garantisce il coinvolgimento precoce del testing lungo l'intero value stream. Un ulteriore obiettivo di quest'area di processo è definire un approccio di test coerente su più livelli di test (che, in un contesto DevOps, sono rappresentati come fasi della pipeline CI/CD), basato sui rischi identificati e sulla strategia di test definita, e fornire un piano di test complessivo, fondato sul ciclo di vita del test definito.

3.3.1 SG1 Istituire gli Asset Organizzativi di Processo di Test

Una responsabilità importante dell'organizzazione di test è definire, documentare e mantenere un processo di test standard, in linea con la policy e gli obiettivi di test dell'organizzazione. Tale processo include, tra l'altro, la descrizione dei vari livelli di test e dei tipi di test da eseguire. Poiché TMMi non richiede processi di test dettagliatamente documentati, l'uso di template arricchiti da esempi rappresenta spesso un approccio efficace per stabilire un modo di lavorare comune [Galen]. Gli Agile Testing Quadrants, nei quali i team concordano i livelli e i tipi di test applicati in un progetto, costituiscono un metodo semplice e di alto livello per rappresentare il processo e la strategia di test concordati. I livelli di test obbligatori vengono quindi riportati come criteri nella Definition of Done. Anche una vista di alto livello delle fasi di testing all'interno della pipeline CI/CD può fungere da valido documento di riferimento comune. Un template identifica le attività necessarie per raccogliere informazioni e documentare i risultati, implicitamente incorporate nella sua struttura. Un processo non deve necessariamente essere una sequenza rigida di attività; laddove esistano dipendenze, queste possono essere indicate tramite note all'interno del template. I template hanno un valore pratico nel comunicare l'intento reale di un processo e aiutano a evitare le ambiguità tipiche dei documenti di processo eccessivamente descrittivi. In un'organizzazione DevOps con processi efficaci, può essere sufficiente documentarli in modo leggero, aggiungendo eventualmente alcuni elementi. In generale, le organizzazioni DevOps fanno ampio affidamento sull'automazione e i processi e le regole sono spesso incorporati e resi obbligatori dagli strumenti. Di conseguenza, anche se i processi di test non sempre esistono come documenti separati, rimangono comunque una parte cruciale del processo di sviluppo e distribuzione. È tuttavia fondamentale che tutti i soggetti coinvolti abbiano una comprensione comune dei processi applicati. Alcune linee guida chiave comunemente utilizzate nelle organizzazioni DevOps quando si definiscono i processi sono le seguenti:

- Gli elementi del processo che costituiscono obblighi ("must do") sono separati dalle linee guida.
- Un processo è generalmente lungo non più di due pagine.
- I processi non contengono informazioni operative di tipo "how-to" né informazioni sugli strumenti, a meno che non si sia deciso di renderle obbligatorie per tutti i progetti, indipendentemente da dimensione e complessità.
- Le linee guida separate contengono le opzioni di tailoring/pianificazione e le informazioni di tipo "how-to". Si noti che, per le informazioni di tipo "how-to", è anche possibile limitarsi a fare riferimento a un libro o a una pubblicazione esistente.

Le grandi organizzazioni possono adottare pratiche di platform engineering, nelle quali i workflow standard degli elementi di backlog o i workflow delle pipeline CI/CD sono resi obbligatori. Ciò significa che i processi definiti sono implementati all'interno di strumenti specifici che devono essere utilizzati dai team. Qualora sia previsto il tailoring, la fase del ciclo di vita può essere opzionale.

In molte grandi organizzazioni non è raro riscontrare quattro livelli di asset di processo, quali policy, processi/pratiche, istruzioni operative/procedure ed elementi abilitanti/template. Questo non è dovuto a requisiti specifici di TMMi, bensì al modo in cui molte grandi organizzazioni hanno scelto di implementare i propri asset di processo. Sebbene la scelta degli asset di processo sia responsabilità di ciascuna organizzazione, molte organizzazioni DevOps stanno riscontrando che due livelli di asset di processo sono sufficienti. Ciò si ottiene consolidando una dichiarazione di policy con la descrizione del processo associato, che racchiude ciò che “deve essere fatto” nell'esecuzione del processo. Il secondo livello contiene le linee guida operative (“how-to”) per l'esecuzione e il tailoring del processo. Questo livello può essere considerato un supporto al tailoring, in quanto conferisce al team autonomia nel decidere come

lavorare nello specifico all'interno del framework definito, e di norma include template di supporto. Nella maggior parte delle organizzazioni DevOps, le procedure passo-passo vengono sostituite da guide agli strumenti e da attività di formazione/mentoring. Come già indicato, un template, ad esempio un test charter template, un test session sheet template o un test plan template, può fungere esso stesso da processo, con le attività di processo necessarie implicitamente incorporate nella struttura del template. Questa è una tecnica comunemente osservata per sviluppare descrizioni di processo Agile efficaci. [McMahon]

L'insieme dei processi di test standard dell'organizzazione può essere adattato (tailored) dai progetti o dai team per creare i propri processi definiti specifici. Nel contesto della pratica specifica SP 1.3 Stabilire criteri e linee guida di tailoring, molte organizzazioni adottano il cosiddetto approccio di “tailor down”. Tale approccio richiede che le persone investano tempo per spiegare perché qualcosa non è necessario. Quando si applica il tailor down, fino a che punto ci si può spingere? Esiste un insieme minimo definito? Se invece si parte da un insieme minimo, dove l'insieme minimo rappresenta ciò che tutti devono fare, si elimina il rischio di escludere tramite tailoring un obbligo (“must do”). Un approccio di “tailoring up” è molto più coerente con gli approcci Agile e DevOps ed è pienamente conforme a TMMi. Un insieme di criteri semplici può aiutare a porre le domande corrette durante il tailoring up. In assenza di criteri di supporto, vi è il rischio che alcuni team estendano eccessivamente il processo.

Per supportare la distribuzione del processo di test standard, viene creata e mantenuta una libreria di asset di processo di test, spesso sotto forma di wiki, nella quale i tester possono trovare template, best practice e checklist a supporto delle attività quotidiane. Di norma, tali template sono parte delle implementazioni degli strumenti, ad esempio come blueprint di pipeline CI/CD, campi obbligatori o template all'interno di uno strumento di gestione degli elementi di backlog. La creazione di una libreria di asset di processo di test ha senso anche in un contesto DevOps, purché i team ottengano un reale valore aggiunto dall'utilizzo di tali asset

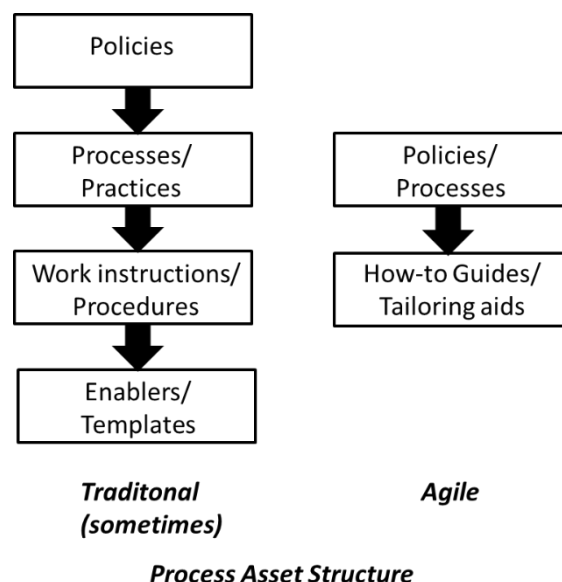


Figura 5: Livelli di risorse di processo

durante le attività di test. Considerazioni analoghe valgono per un database di processo di test, nel quale vengono raccolti e resi disponibili ai team dati di test, quali stime, copertura e misure di qualità. È importante garantire che ciò che viene raccolto e reso disponibile favorisca l'apprendimento trasversale tra team e la condivisione delle esperienze, apportando valore ai team stessi.

Naturalmente, anche in un contesto DevOps vengono utilizzati standard per l'ambiente di lavoro, al fine di guidare la creazione degli ambienti di progetto.

3.3.2 SG2 Integrare i Modelli di Ciclo di Vita del Test con i Modelli di Sviluppo

Il modello standard del ciclo di vita del test definisce le principali attività e i deliverable per i vari livelli di test. Tale modello è allineato al modello di ciclo di vita dello sviluppo, al fine di integrare le attività di testing in termini di fasi, milestone, deliverable e attività. L'integrazione dei cicli di vita è realizzata in modo tale da garantire il coinvolgimento precoce del testing nei progetti per ogni elemento di backlog. In un contesto DevOps, sviluppo e testing dovrebbero essere completamente integrati nel ciclo di vita e il testing dovrebbe iniziare il prima possibile. Poiché i tester fanno parte del team e partecipano alla pianificazione delle iterazioni, i rischi vengono identificati anticipatamente e le attività di test concordate; ciò influisce su stime e impegni, garantendo l'integrazione del testing nel ciclo di vita.

Questo specifico obiettivo è quindi applicabile anche a un'organizzazione che adotta lo sviluppo software DevOps. In un contesto DevOps, sviluppo e testing dovrebbero già essere pienamente integrati a livello di team e dovrebbe esistere una comprensione condivisa delle modalità con cui ciò avviene. In tali casi, non è necessario intraprendere ulteriori azioni. In altre situazioni, in cui l'integrazione e il coinvolgimento precoce non sono ancora pienamente implementati, questo obiettivo rappresenta un'area di miglioramento molto rilevante e dovrebbe condurre a un allineamento decisamente migliore tra le attività di sviluppo e di test svolte nell'ambito di un'iterazione.

3.3.3 SG3 Istituire un Master Test Plan

Al livello 3 di TMMi, il testing comprende la pianificazione di test a livello master, che affronta il coordinamento delle attività di test, delle responsabilità e dell'approccio di test su tutti i livelli di test. Ciò consente di evitare ridondanze o omissioni non necessarie tra i vari livelli di test e può aumentare in modo significativo l'efficienza e la qualità complessiva del processo di test. Il master test plan descrive l'applicazione della strategia di test per uno specifico progetto, includendo i livelli di test da eseguire e le relazioni tra di essi. Al livello 2 di TMMi, la pianificazione dei test è già affrontata sia a livello di release sia di iterazione come parte dell'area di processo Test Planning. Un team DevOps esegue tipicamente più livelli di test, che dovrebbero essere adeguatamente considerati nella pianificazione delle release e delle iterazioni.

Tuttavia, alcuni progetti che operano secondo un modello di ciclo di vita DevOps hanno adottato un approccio parzialmente ibrido, in cui, oltre al testing svolto all'interno dei team DevOps, alcune attività di test sono organizzate al di fuori dei team. In tali circostanze, un master test plan definisce e gestisce la relazione tra i team DevOps e i livelli o tipi di test svolti al di fuori del loro ambito, ad esempio test non funzionali esternalizzati (sicurezza, performance) o test di integrazione di sistema con parti esterne. Questo è solitamente rappresentato come una fase aggiuntiva sulla task board degli elementi di backlog. In tali casi, le pratiche specifiche associate a questo obiettivo risultano generalmente applicabili.

Nel caso in cui tutto il testing venga svolto all'interno dei team DevOps, un master test plan specifico non apporta valore aggiunto e questo obiettivo TMMi risulta probabilmente non rilevante. In tali situazioni, la

pianificazione a livello di release e di iterazione è considerata sufficiente a coprire tutti gli aspetti di testing necessari. Tuttavia, è utile verificare che il team disponga di un approccio di test generico, condiviso e continuamente migliorato. La Definition of Done, le pipeline o i workflow CI/CD, oppure un semplice documento basato sugli Agile Testing Quadrants possono essere considerati come una forma leggera di master test plan.

3.4 Area di Processo 3.4 Non-Functional Testing

Lo scopo dell'area di processo Testing Non Funzionale è migliorare le capacità del processo di test includendo il testing non funzionale durante la pianificazione, la progettazione e l'esecuzione dei test.

L'importanza del testing non funzionale è indipendente dal ciclo di vita adottato. Il prodotto in fase di sviluppo determina quali aspetti non funzionali siano rilevanti per il testing, e ciò vale anche per lo sviluppo DevOps. Il testing non funzionale è pertanto applicabile anche allo sviluppo software DevOps.

Tuttavia, a seconda della natura degli aspetti non funzionali e dell'approccio di test adottato, alcune caratteristiche non funzionali, come prestazioni e affidabilità, non possono essere testate efficacemente in una breve iterazione utilizzando approcci tradizionali. È quindi spesso necessario un approccio differente al testing degli attributi non funzionali. Uno dei cambiamenti fondamentali introdotti da DevOps è che gli attributi di qualità vengono testati precocemente e non rimandati alle fasi finali. Sebbene non tutti i componenti o le funzionalità siano disponibili fin dalla prima iterazione per eseguire un testing non funzionale completo, i risultati ottenuti possono comunque fornire indicazioni precoci su potenziali problemi di qualità.

Le user story dovrebbero includere sia elementi funzionali sia non funzionali per garantire lo sviluppo del prodotto corretto per utenti e clienti. Le caratteristiche di qualità definite dalla norma ISO 25010 possono supportare la strutturazione dei requisiti e i tester dovrebbero considerare tali elementi non funzionali [ISO 25010]. In un contesto DevOps, oltre alle caratteristiche di qualità ISO 25010, viene introdotta anche l'osservabilità, supportata dalla telemetria¹, come caratteristica di qualità.

3.4.1 SG1 Eseguire una Valutazione dei Rischi di Prodotto Non Funzionali

Le sessioni e le tecniche di valutazione del rischio per i progetti DevOps, quali risk poker, ROAM e threat modelling, già identificate e descritte nell'obiettivo SG1 Eseguire la valutazione del rischio dell'area di processo Test Planning, vengono ora esplicitamente estese per includere anche aspetti e rischi non funzionali. In DevOps vi è spesso un focus particolare su osservabilità, sicurezza e conformità. Con la crescente digitalizzazione e l'adozione del cloud, i sistemi IT affrontano rischi e vulnerabilità crescenti. Il threat modelling è un processo svolto nelle fasi iniziali del ciclo di vita delle funzionalità, durante l'analisi e il refinement, in cui viene condotta la valutazione del rischio per identificare le minacce e mitigarle tramite un modello strutturato. I risultati si

¹ L'osservabilità si riferisce alla capacità di comprendere lo stato di un componente o di un sistema analizzandone gli output esterni, al fine di consentire il monitoraggio, la risoluzione dei problemi e l'ottimizzazione. L'osservabilità utilizza logging, tracing e metriche per fornire informazioni e creare una visione coerente dello stato del software.

Le informazioni vengono raccolte automaticamente tramite la telemetria. La telemetria raccoglie misurazioni relative alle caratteristiche di qualità funzionali e non funzionali. Esempi di telemetria sono la telemetria del sistema in esercizio (live site telemetry), utilizzata per misurare il funzionamento del software, ad esempio l'utilizzo della memoria o il carico della CPU, e la telemetria utente, utilizzata per misurare l'interazione degli utenti, ad esempio il tracciamento dei clic per l'uso delle funzionalità o il monitoraggio del tasso di completamento dei moduli.

riflettono spesso in criteri di accettazione non funzionali a livello di funzionalità e possono guidare la creazione di ulteriori user story o decisioni architetturali dedicate. A livello di iterazione, la valutazione del rischio viene effettuata sulle user story durante il refinement, utilizzando i requisiti non funzionali come input principale e aggiornando di conseguenza i criteri di accettazione. Idealmente, alle sessioni di valutazione del rischio partecipano tutti i membri del team, incluso il product owner e, se necessario, altri stakeholder. Per alcune aree non funzionali può essere richiesto il supporto di specialisti. Le sessioni di valutazione del rischio producono tipicamente un elenco documentato e prioritizzato dei rischi non funzionali di prodotto, aggiornando la documentazione di funzionalità e user story. Tale elenco non solo guida la progettazione della pipeline CI/CD e l'approccio al testing non funzionale, ma contribuisce anche a modellare l'architettura del prodotto e le tecnologie selezionate per lo sviluppo, ad esempio l'uso della containerizzazione per garantire scalabilità e affidabilità. Come indicato nell'area di processo Test Planning, nei progetti DevOps sia il processo adottato sia la documentazione risultante sono generalmente molto più snelli rispetto a quelli di un progetto tradizionale con ciclo di vita sequenziale.

3.4.2 SG2 Definire un Approccio di Test Non Funzionale

Viene definito un approccio di test per le caratteristiche di qualità non funzionali rilevanti, al fine di mitigare i rischi di prodotto non funzionali identificati e prioritizzati. La pipeline CI/CD deve essere progettata in modo da includere anche il testing non funzionale. Per una specifica iterazione, ulteriori aspetti non funzionali da testare vengono identificati durante la pianificazione dell'iterazione, mentre i test non funzionali presenti nella pipeline CI/CD vengono eseguiti per ogni modifica. Le caratteristiche di qualità non funzionali vengono inoltre misurate dopo il deployment in produzione, utilizzando la telemetria a supporto dell'osservabilità. In DevOps, gli Service Level Indicator (SLI) e gli Service Level Objective (SLO) rappresentano un approccio tipico per la gestione di tali misurazioni. L'elenco prioritizzato degli aspetti non funzionali da testare è generalmente correlato alle user story sviluppate e testate nell'iterazione. Durante l'iterazione possono emergere nuovi rischi di prodotto non funzionali che richiedono ulteriori attività di testing; tali tematiche vengono solitamente discusse durante i daily stand-up meeting.

L'approccio di test non funzionale definito a livello di iterazione per mitigare i rischi non funzionali include tipicamente l'identificazione di metodi e tecniche di test non funzionali appropriati, in funzione del livello e del tipo di rischio. Di norma, include anche l'utilizzo di strumenti di supporto e l'approccio all'automazione del testing non funzionale. L'approccio di test non funzionale a livello di release è generalmente definito a un livello più alto ed è basato sulla pipeline CI/CD definita. Sia l'approccio di test non funzionale a livello di release sia quello a livello di iterazione fanno parte dell'approccio di test complessivo e sono documentati o resi disponibili sul wiki del team o del progetto. I risultati dei test non funzionali devono essere presentati su dashboard di prodotto.

I criteri di uscita non funzionali fanno parte della Definition of Done (DoD). È importante che la DoD includa criteri specifici relativi al testing non funzionale, ad esempio tempi di risposta delle API o requisiti di sicurezza come "le pagine web front-end sono state testate rispetto alla OWASP Top 10". Gli SLI e gli SLO misurati dopo il deployment come parte della pipeline CI/CD fungono anch'essi da criteri di uscita, con l'aspettativa tipica di evitare il degrado degli SLI o degli SLO. L'iterazione dovrebbe portare all'implementazione dell'insieme concordato di criteri di accettazione non funzionali ed eventualmente di user story dedicate, soddisfacendo i criteri di uscita (di test) non funzionali definiti nella DoD. Oltre alla DoD a livello di iterazione, spesso esiste anche una DoD a livello di funzionalità o di release, che si estende su più iterazioni. Inoltre, in un contesto DevOps vi è una maggiore enfasi sugli approcci di Shift Right, che prevedono la misurazione delle

caratteristiche non funzionali in produzione tramite monitoraggio continuo, utilizzando SLI e SLO dopo il deployment.

3.4.3 SG3 Eseguire l'Analisi e la Progettazione dei Test Non Funzionali

Questo specifico obiettivo segue in larga misura le stesse pratiche dell'obiettivo SG1 Eseguire l'analisi e la progettazione dei test utilizzando tecniche di progettazione dei test dell'area di processo Test Design and Execution, ma applicate alla prospettiva non funzionale. Durante l'analisi e la progettazione dei test, l'approccio di test non funzionale viene tradotto in condizioni di test e test concreti. In DevOps, analisi e progettazione dei test ed esecuzione dei test sono attività che si supportano reciprocamente e che generalmente procedono in parallelo durante l'iterazione. Ciò vale anche per la maggior parte del testing non funzionale. L'analisi dei test non funzionali non è pertanto un'attività separata ed esplicita, ma piuttosto un'attività implicita svolta dai tester (DevOps quality engineer) come parte del loro ruolo all'interno dello sviluppo collaborativo e del refinement delle user story.

Sulla base dell'analisi dei criteri di accettazione non funzionali e delle user story, vengono identificate le condizioni di test non funzionali. Le condizioni di test rappresentano essenzialmente l'identificazione degli aspetti che devono essere testati/coperti. Se i criteri di accettazione definiti sono sufficientemente dettagliati e chiari, essi tendono a sostituire il ruolo delle condizioni di test tradizionali. I criteri di accettazione vengono quindi tradotti in test non funzionali. Nel testing non funzionale è spesso vantaggioso effettuare l'analisi dei test a un livello più elevato rispetto alle singole user story, ad esempio analizzando una funzionalità, un epic o un insieme di user story, per identificare condizioni di test non funzionali più astratte che si estendono su più user story. Con l'applicazione del principio test-first in DevOps, i test non funzionali che coprono l'insieme delle condizioni di test non funzionali vengono identificati (ed eventualmente automatizzati) prima, o almeno in parallelo, allo sviluppo del codice.

Per la maggior parte del testing non funzionale manuale, i test vengono identificati e affinati man mano che il team procede con l'esecuzione dei test non funzionali. I test non sono generalmente documentati in modo dettagliato come nei progetti tradizionali, ma piuttosto sotto forma di idee di test, ad esempio nel contesto del testing esplorativo. Per aree non funzionali complesse e critiche, un approccio più tradizionale alla progettazione dei test e allo sviluppo dei casi di test, basato su tecniche formali di progettazione dei test non funzionali, può rappresentare il modo migliore per coprire i rischi. Anche in questo caso, tuttavia, il livello di documentazione rimane limitato rispetto al testing non funzionale in un ambiente tradizionale con ciclo di vita sequenziale. La prioritizzazione dei test non funzionali segue tipicamente la prioritizzazione della user story che essi coprono; tuttavia, la priorità può essere influenzata anche dal tempo necessario per preparare ed eseguire uno specifico test non funzionale.

L'identificazione dei requisiti e delle user story relativi a osservabilità e telemetria può essere anch'essa un risultato dell'analisi e progettazione non funzionale, al fine di consentire la misurazione degli aspetti non funzionali in produzione.

Vengono identificati i dati di test specifici necessari a supportare l'esecuzione dei test non funzionali. In DevOps, i dati di test necessari non sono generalmente specificati in anticipo in un documento di specifica dei test, ma sono spesso registrati come parte della user story, ad esempio sotto forma di persona. Tuttavia, se gli strumenti e/o le funzionalità necessarie sono disponibili, i dati di test richiesti per supportare i test non funzionali vengono spesso creati immediatamente per consentire un rapido avvio dell'esecuzione dei test non funzionali manuali. Al contrario, per i test non funzionali automatizzati, i dati devono tipicamente essere

definiti in anticipo. Inoltre, poiché alcuni requisiti non funzionali vengono testati in produzione, i dati di test possono essere dati di produzione; in tali casi, gli aspetti di privacy e GDPR sono fondamentali.

È necessario stabilire e mantenere la tracciabilità tra requisiti non funzionali, condizioni di test e test. I team devono dimostrare di aver coperto le diverse user story non funzionali e i relativi criteri di accettazione come parte delle attività di testing. Nella maggior parte delle organizzazioni DevOps, le user story costituiscono la base per lo sviluppo di un insieme di criteri di accettazione correlati e dei relativi test. Utilizzando questo approccio, è possibile ottenere una tracciabilità orizzontale dai requisiti ai test.

3.4.4 SG4 Eseguire l'Implementazione dei Test Non Funzionali

L'implementazione dei test riguarda la predisposizione di tutto ciò che è necessario per avviare l'esecuzione dei test. Tipicamente, lo sviluppo della documentazione di test, ad esempio le procedure di test a supporto dell'esecuzione, è ridotto al minimo. Vengono invece sviluppati e prioritizzati script di test automatizzati (in particolare di regressione). In DevOps, tutto ciò che può essere automatizzato dovrebbe essere automatizzato.

L'implementazione dei test non funzionali segue molte delle pratiche già descritte nell'obiettivo SG2 Eseguire l'implementazione dei test dell'area di processo Test Design and Execution. Le attività specifiche e le modalità di implementazione dei test non funzionali dipendono in larga misura dall'approccio definito, dalle tecniche utilizzate e dalle caratteristiche non funzionali da testare e dal livello a cui devono essere verificate. Ad esempio, il testing esplorativo, che richiede una preparazione ridotta, può essere adatto al testing di usabilità, ma è spesso meno indicato per test estesi di affidabilità e performance.

Per alcune caratteristiche di qualità non funzionali, la disponibilità dei dati di test è essenziale e tali dati devono essere creati durante l'implementazione dei test. Questo aspetto è in gran parte analogo a quanto avviene nei progetti tradizionali, sebbene i dati possano essere documentati in modo più sintetico, consentendone comunque il riutilizzo per il testing di regressione. Come già indicato, poiché alcuni requisiti non funzionali vengono testati in produzione, i dati di test possono essere dati di produzione.

Naturalmente, l'implementazione e la preparazione dei test iniziano il prima possibile, in parallelo alle altre attività (di testing). Non si tratta di una fase separata, ma di un insieme di attività (tipicamente elencate sulla task board) necessarie per consentire un'esecuzione dei test efficiente ed efficace.

3.4.5 SG5 Svolgere l'Esecuzione dei Test Non Funzionali

Come per l'obiettivo precedente di quest'area di processo, si fa in larga misura riferimento all'obiettivo SG3 Eseguire l'esecuzione dei test dell'area di processo Test Design and Execution, discussa in precedenza in questo documento. Le pratiche di esecuzione dei test non funzionali, di segnalazione dei difetti e di redazione dei log di test in un ambiente DevOps sono sostanzialmente le stesse del testing funzionale in un contesto DevOps. Esse sono generalmente caratterizzate da un livello di documentazione molto inferiore rispetto ai progetti tradizionali. Spesso non vengono prodotti né procedure di test dettagliate né log di test completi. Si noti che gli aspetti non funzionali vengono anche testati e misurati dopo il deployment, utilizzando Service Level Indicator (SLI) e Service Level Objective (SLO) come modalità tipica di gestione.

L'esecuzione dei test non funzionali è parte integrante della pipeline CI/CD ed è svolta in linea con le priorità definite durante la pianificazione dell'iterazione. Alcuni test possono essere eseguiti utilizzando procedure di test documentate, ma nella maggior parte dei casi i test non funzionali vengono eseguiti utilizzando il testing esplorativo o basato su sessioni come framework di riferimento. Con lo sviluppo iterativo aumenta l'esigenza

di organizzare e strutturare il testing di regressione, che viene realizzato mediante test di regressione automatizzati e strumenti di supporto. Il testing di regressione si applica naturalmente anche agli aspetti non funzionali del sistema ritenuti rilevanti.

I difetti non funzionali individuati durante il testing possono essere registrati e segnalati dal team. Nei progetti DevOps è frequente il dibattito sull'opportunità di registrare tutti i difetti individuati. Alcuni team registrano solo i difetti che sfuggono all'iterazione, altri quelli che non possono essere corretti immediatamente, altri ancora solo i difetti ad alta priorità. Qualora non vengano registrati tutti i difetti individuati, devono essere definiti criteri chiari per stabilire quali difetti debbano essere registrati e quali no. Talvolta i difetti vengono visualizzati sulla task board, ad esempio come adesivi su un'attività o come attività separate, per evidenziare che stanno bloccando il completamento di una user story e delle relative attività. Alcuni team scelgono di registrare e documentare i difetti utilizzando strumenti dedicati, come sistemi di gestione o tracciamento dei difetti. Tali strumenti dovrebbero essere utilizzati in modo il più possibile snello, evitando di imporre la compilazione di elementi dei report di difetto non funzionale che apportano poco o nessun valore aggiunto. Nei casi in cui i requisiti non funzionali vengano misurati in produzione, un indicatore di degrado (ad esempio un SLI che non soddisfa l'SLO) può attivare un rollback automatico a una versione precedente, mantenendo le prestazioni complessive del sistema.

È considerata una buona pratica, anche nei team DevOps, registrare dati durante l'esecuzione dei test non funzionali per determinare se gli elementi testati soddisfano i criteri di accettazione definiti e possano essere effettivamente considerati "done". Le informazioni registrate dovrebbero essere raccolte e/o sintetizzate in uno strumento di gestione dello stato (ad esempio strumenti di test management, strumenti di task management o task board), in modo da rendere facilmente comprensibile al team e agli stakeholder lo stato corrente di tutte le attività di testing svolte.

3.5 Area di Processo 3.5 Peer Reviews

Lo scopo dell'area di processo Peer Review è verificare che i prodotti di lavoro soddisfino i requisiti specificati e rimuovere i difetti dai prodotti di lavoro selezionati in modo tempestivo ed efficiente. Un importante effetto collaterale è lo sviluppo di una migliore comprensione dei prodotti di lavoro e dei difetti che potrebbero essere prevenuti.

3.5.1 SG1 Definire un Approccio di Peer Review

I team DevOps generalmente non eseguono peer review formali nel senso tradizionale, ossia non prevedono un singolo momento specifico in cui le persone si riuniscono per fornire feedback su un prodotto. Tuttavia, essi raggiungono l'obiettivo delle peer review attraverso peer review continue e meno formali durante tutto lo sviluppo. Questa è una pratica comune in molte organizzazioni DevOps. Poiché DevOps pone un forte accento sul flusso e sull'adozione di pratiche lean nello sviluppo software, qualsiasi pratica di revisione deve supportare questo sviluppo rapido, evitando di creare colli di bottiglia o ritardi nel flusso delle attività. È tipico esaminare elementi di lavoro di dimensioni ridotte, ad esempio mediante frequenti code review su poche righe di codice o attraverso la definizione collaborativa dei criteri di accettazione degli elementi di backlog. Le pratiche di revisione sono considerate così importanti da essere rese inevitabili nel processo, ad esempio rendendo obbligatoria una revisione per ogni pull request o imponendo la conformità alla Definition of Ready. Questo obiettivo TMMi copre le pratiche per definire un approccio di peer review all'interno di un progetto. Un approccio di revisione definisce come, dove e quando devono svolgersi le attività di revisione e se tali attività siano formali o informali. La definizione di un approccio di peer review è applicabile anche ai progetti

DevOps. Per quanto possibile, parti dell'approccio di revisione vengono stabilite e rese obbligatorie implementandole direttamente negli strumenti, ad esempio tra le fasi della task board che gestisce il flusso di lavoro, nelle pull request o come quality gate nella pipeline CI/CD.

Esempi di peer review tipicamente svolte nei progetti DevOps includono:

- sessioni di backlog refinement sulle user story con il team e gli stakeholder di business, svolte regolarmente durante l'iterazione;
- incontri quotidiani con altri membri del team per discutere i prodotti di lavoro in fase di sviluppo, ad esempio codice o test, e fornire feedback;
- dimostrazioni frequenti e anticipate dei prodotti ai clienti, almeno al termine di ogni iterazione durante la iteration review;
- revisioni di prodotti di lavoro quali user story, modifiche al codice tramite pull request, script e file di configurazione nell'ambito dell'Infrastructure-as-Code;
- lavoro in coppia su qualsiasi attività durante l'iterazione.

Specifiche di scarsa qualità sono spesso una delle principali cause di fallimento dei progetti. I problemi di specifica possono derivare dalla scarsa comprensione da parte degli utenti delle proprie reali esigenze, dall'assenza di una visione globale del sistema, da funzionalità ridondanti o contraddittorie e da altre forme di cattiva comunicazione. Nello sviluppo DevOps, le user story vengono redatte per catturare i requisiti dal punto di vista dei rappresentanti di business, degli sviluppatori e dei tester. Nello sviluppo sequenziale, questa visione condivisa di una funzionalità viene raggiunta tramite revisioni formali dopo la redazione dei requisiti; nello sviluppo DevOps, essa viene raggiunta attraverso revisioni informali frequenti durante la definizione dei requisiti. Queste sessioni di revisione informali sono spesso denominate backlog refinement session. Durante tali incontri, il team e il rappresentante di business utilizzano tecniche di revisione per stabilire una comprensione comune, individuare il livello di dettaglio necessario per l'implementazione e chiarire le questioni aperte.

La revisione quasi continua dei prodotti di lavoro in fase di sviluppo da parte del team è parte integrante dell'approccio "whole team". Tali revisioni sono svolte con l'obiettivo di individuare precocemente i difetti e di identificare opportunità di miglioramento. Metodi e tecniche adottati in DevOps, come XP o il lavoro in coppia, includono le peer review come pratica fondamentale per creare cicli di feedback per i team. Naturalmente, in presenza di elevata complessità o rischio, il team può decidere di applicare una tecnica di revisione semi-formale o formale, ad esempio un'ispezione. In questi casi, esiste una motivazione chiara per investire maggiori sforzi in una revisione più strutturata e adottare un modo di lavorare più disciplinato. Nell'ambito dell'approccio di revisione possono essere definiti criteri che stabiliscono quando applicare una tecnica di revisione più formale.

Mentre il requirements engineering tradizionale enfatizza la validazione dei requisiti tramite metodi quali revisioni informali, walkthrough, ispezioni o lettura basata su prospettive, i metodi Agile e DevOps mirano piuttosto a validare i requisiti attraverso feedback frequenti e precoci, al fine di implementare rapidamente incrementi di prodotto di valore. La necessità di una validazione formale precoce è ridotta mostrando rapidamente risultati concreti, sotto forma di prototipi o incrementi di prodotto integrati e distribuiti in produzione. Se un incremento non soddisfa pienamente i requisiti degli stakeholder, la differenza viene reintrodotta nel product backlog sotto forma di nuovi requisiti e prioritizzata insieme agli altri elementi di backlog. La dimostrazione di quanto effettivamente realizzato durante un'iterazione, svolta nel corso della iteration review, rappresenta un modo estremamente efficace per stimolare una conversazione di validazione basata su qualcosa di concreto. Nulla focalizza il confronto quanto la possibilità di vedere realmente come

funziona una soluzione e di comprendere l'impatto delle funzionalità distribuite sul comportamento degli utenti finali. La dimostrazione è un'attività svolta durante la iteration review: le funzionalità integrate nel sistema vengono mostrate e discusse con gli stakeholder e le necessarie modifiche vengono apportate al product backlog o al piano di release per riflettere le nuove conoscenze emerse dalla discussione.

I criteri di peer review possono essere definiti in modo automatizzato o come checklist all'interno del workflow. Nel caso delle code review, vengono definiti criteri di ingresso quali l'esecuzione obbligatoria dei test unitari, misure di complessità del codice e conformità alle linee guida di stile del codice. Ciò garantisce che solo il codice conforme ai criteri di ingresso venga sottoposto a revisione. Come criterio di uscita, il codice deve aver superato la revisione, di norma da parte di più membri del team. In caso contrario, la revisione deve essere ripetuta dopo l'aggiornamento del codice, il che implica generalmente che il codice non possa essere integrato (merge).

Per gli elementi di backlog, i criteri di ingresso sono spesso applicabili solo alle revisioni formali e risultano quindi generalmente non rilevanti per le sessioni di refinement nei progetti DevOps. Tuttavia, l'utilizzo di criteri di uscita è applicabile e apporta valore. INVEST [Wake] è un esempio di insieme di criteri di uscita comunemente utilizzato nei progetti DevOps durante la revisione e l'aggiornamento delle user story. INVEST è un acronimo che include i seguenti requisiti che caratterizzano una buona user story:

- Independent
- Negotiable
- Valuable
- Estimable
- Small
- Testable

Naturalmente, esistono anche altri criteri di uscita legati alla qualità che possono essere utilizzati con beneficio.

I criteri di uscita delle revisioni sono definiti nella Definition of Done per i diversi prodotti di lavoro. Gli elementi che non soddisfano tali criteri vengono riportati indietro e non possono avanzare allo stadio successivo.

3.5.2 SG2 Eseguire le Peer Review

Come qualsiasi approccio, non è sufficiente che esso sia definito e condiviso: deve anche essere applicato. Il team deve dedicare una quantità significativa di tempo alle sessioni di backlog refinement durante l'iterazione, applicando revisioni informali (ed eventualmente formali) come parte della routine quotidiana, e deve prevedere dimostrazioni regolari agli stakeholder. Le dimostrazioni a stakeholder e clienti sono attese almeno al termine di ogni iterazione.

È importante che il tester, in quanto membro del team DevOps, partecipi alle sessioni di revisione. Ciò è particolarmente rilevante per le sessioni in cui vengono discussi e revisionati prodotti di lavoro che costituiscono la base per le attività di testing durante l'iterazione, ad esempio nelle sessioni di backlog refinement e nella progettazione collaborativa. La prospettiva unica del tester contribuisce tipicamente a migliorare la user story, individuando dettagli mancanti o requisiti non funzionali. Il tester può supportare in modo particolare l'identificazione e la definizione dei criteri di accettazione di una user story. Tecniche come BDD e ATDD possono essere utilizzate per definire i criteri di accettazione sotto forma di test di accettazione, consentendo di determinare se la user story soddisfa il criterio di testabilità. In questo modo, il tester

contribuisce ponendo ai rappresentanti di business domande esplorative del tipo “e se...?”, proponendo modalità di test della user story e confermando i criteri di accettazione.

La pratica specifica 2.3 Analizzare i dati delle peer review è particolarmente rilevante nel caso delle revisioni formali. Nelle revisioni formali viene investito un notevole sforzo ed è quindi importante garantire che tale sforzo sia efficace ed efficiente; a tal fine, i dati delle peer review vengono raccolti e comunicati al team per apprendere e ottimizzare il processo di revisione. Come già indicato, le revisioni formali sono meno comuni nei progetti DevOps. Mentre la raccolta dei dati è un elemento essenziale delle revisioni formali, essa è molto meno frequente nelle revisioni informali. Nel decidere se raccogliere dati durante le peer review, è opportuno porsi le seguenti domande:

- chi utilizzerà questi dati, se vengono raccolti?
- in che modo questi dati sono collegati agli obiettivi di business?

Se nessuno utilizzerà i dati in modo significativo, è preferibile non sprecare risorse preziose per raccogliarli. Tuttavia, in DevOps vengono tipicamente utilizzati strumenti che raccolgono automaticamente i risultati e i dati delle revisioni e li rendono disponibili sotto forma di informazioni su dashboard, rendendo il processo molto più efficiente. Inoltre, durante le retrospettive, quando il team analizza i dati e i risultati delle revisioni per comprendere le cause radice e individuare opportunità di miglioramento, tali informazioni assumono un valore elevato.

Glossario dei termini specifici DevOps

Blue-green deployment	Strategia di distribuzione in cui vengono mantenuti due ambienti di produzione identici, uno attivo e uno inattivo, per testare una nuova versione nell'ambiente inattivo prima di trasferire tutto il traffico dall'ambiente attivo una volta verificata.
Canary testing	Approccio di test in cui le modifiche vengono rilasciate a un piccolo sottoinsieme di utenti, consentendo la validazione prima di un rilascio incrementale più ampio.
CI/CD pipeline	Sequenza obbligatoria di fasi necessarie per distribuire una nuova versione del software.
Containerizzazione	Forma leggera di virtualizzazione che incapsula un'applicazione e le sue dipendenze in un'unica unità eseguibile chiamata container.
Continuous delivery (CD)	Procedura automatizzata di sviluppo software in cui le modifiche al codice vengono automaticamente compilate, testate e rese distribuibili in produzione.
Continuous deployment (CD)	Procedura automatizzata di rilascio software in cui le modifiche al codice che superano tutti i test vengono automaticamente distribuite in produzione senza intervento manuale.
Continuous monitoring	Raccolta automatizzata di indicatori di prestazione per comprendere in tempo reale il comportamento degli utenti e del sistema.
Dark launch	Strategia di distribuzione in cui una nuova funzionalità o prodotto viene rilasciato senza annuncio pubblico, 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 per sviluppare e gestire efficacemente una soluzione.
Feedback	Scambio continuo di informazioni finalizzato al miglioramento di prodotti, servizi e processi.
Feature toggle	Meccanismo che consente di modificare il comportamento di un sistema senza apportare modifiche al codice.
Flusso	Movimento fluido, lineare e rapido dei prodotti di lavoro da una fase all'altra all'interno di un value stream rilevante.
Frequenza di deployment	Velocità con cui le versioni software vengono rilasciate in produzione.
Infrastructure as Code (IaC)	Pratica di gestione e provisioning dell'infrastruttura IT mediante file di configurazione leggibili da macchina, applicando ad essa le pratiche dello sviluppo software.

Osservabilità	Capacità di comprendere lo stato di un componente o di un sistema analizzandone gli output esterni, al fine di consentire monitoraggio, troubleshooting e ottimizzazione.
Percentuale di modifiche fallite	Proporzione delle modifiche distribuite che causano un fallimento.
Pull Request (PR)	Meccanismo che consente di notificare al team le modifiche apportate al codice, proponendone la revisione, la discussione e l'integrazione nel repository principale.
Service Level Indicator (SLI)	Misura quantitativa dell'affidabilità di un servizio.
Service Level Objective (SLO)	Obiettivo di affidabilità associato a uno specifico service level indicator.
Site Reliability Engineering (SRE)	Approccio che applica pratiche e principi dello sviluppo software all'infrastruttura e alle operazioni, bilanciando l'affidabilità del servizio con la velocità di sviluppo di nuove funzionalità tramite cicli di feedback automatizzati.
Team DevOps multifunzionale	Gruppo di persone con insiemi di conoscenze, competenze e capacità diversi ma parzialmente sovrapposti, che collaborano per il raggiungimento di un obiettivo comune.
Tempo di lead della modifica	Tempo necessario affinché un commit raggiunga l'ambiente di produzione.
Tempo di ripristino dopo un deployment fallito	Durata necessaria per ripristinare il funzionamento normale del sistema dopo un deployment fallito.
Value stream	Sequenza end-to-end di attività necessarie per fornire un prodotto o servizio a un cliente, includendo il flusso di informazioni e materiali.
Virtualizzazione	Simulazione del software e/o dell'hardware su cui viene eseguito altro software.

Riferimenti

- [Baah] A. Baah (2017), Agile quality assurance, Bookbaby
- [Beyer] B. Beyer, C. Jones, J. Petoff and N.R. Murphy (2016). Site Reliability Engineering: How Google Runs Production Systems, 1st. O'Reilly Media, Inc.
- [Cohn] M. Cohn (2009), Succeeding with Agile: Software Development using Scrum, Addison-Wesley
- [DORA16] A. Brown, N. Forsgren, J. Humble, N. Kersten and G. Kim (2016), 2016 State of DevOps Report, Puppet Labs, DORA (DevOps Research & Assessment]
- [DORA24] DORA [2024], DORA's software delivery metrics: the four keys, <https://dora.dev/guides/dora-metrics-four-keys/>, DORA (DevOps Research & Assessment]
- [Galen] R. Galen (2015), Three Pillars of Agile Quality and Testing: Achieving Balanced Results in your Journey Towards Agile Quality, RGCG, LLC
- [ISO25010] ISO/IEC 25010 (2023), Systems and Software Engineering – Systems and Software Quality Requirements and Evaluation (SQuaRE) – Product quality model, International Organization for Standardization
- [McMahon] Paul. E. McMahon (2011), Integrating CMMI with Agile Development – Case Studies and Proven Techniques for Faster Performance Improvement, Addison Wesley
- [Veenendaal12] E. van Veenendaal (2012), PRISMA: Product Risk Assessment for Agile Projects, in: Testing Experience, Issue 04/12, December 2012
- [Veenendaal24] Erik van Veenendaal, Rex Black, and Dorothy Graham (2024), Foundations of Software Testing - ISTQB Certification (5th edition), Cengage
- [Wake] B. Wake (2003), INVEST in Good Stories, and SMART Tasks, xp123.com/articles/invest-in-good-stories-and-smart-tasks