Certified Tester

Syllabus Agile Tester Foundation Level

Versione 2014 (versione italiana 02 aggiornata a febbraio 2022)

International Software Testing Qualifications Board

ITAlian Software Testing Qualifications Board

Foundation Level Extension Agile Tester Working Group: Rex Black (Chair), Bertrand Cornanguer (Vice Chair), Gerry Coleman (Learning Objectives Lead), Debra Friedenberg (Exam Lead), Alon Linetzki (Business Outcomes and Marketing Lead), Tauhida Parveen (Editor), and Leo van der Aalst (Development Lead).

Autori: Rex Black, Anders Claesson, Gerry Coleman, Bertrand Cornanguer, Istvan Forgacs, Alon Linetzki, Tilo Linz, Leo van der Aalst, Marie Walsh e Stephan Weber.

Revisori interni: Mette Bruhn-Pedersen, Christopher Clements, Alessandro Collino, Debra Friedenberg, Kari Kakkonen, Beata Karpinska, Sammy Kolluru, Jennifer Leger, Thomas Mueller, Tuula Pääkkönen, Meile Posthuma, Gabor Puhalla, Lloyd Roden, Marko Rytkönen, Monika Stoecklein-Olsen, Robert Treffny, Chris Van Bael e Erik van Veenendaal; 2013-2014.

Storico delle Revisioni

Versione	Data	Note
Syllabus v0.1	26/07/2013	versione iniziale
Syllabus v0.2	16/09/2013	Commenti della review di WG incorporate alla v01
Syllabus v0.3	20/10/2013	Commenti della review di WG incorporate alla v02
Syllabus v0.7	16/12/2013	Commenti della review alpha incorporate alla v03
Syllabus v0.71	20/12/2013	Aggiornamenti del Working group alla v07
Syllabus v0.9	30/01/2014	Versione Beta
Syllabus 2014	31/05/2014	Versione GA

Indice

Ringraziamenti

Questo documento è stato prodotto da un team dedicato appartenente al gruppo di lavoro dell'International Software Testing Qualifications Board Foundation Level.

Il team per l'estensione Agile desidera ringraziare il gruppo dei revisori e tutte le organizzazioni nazionali per i suggerimenti e gli input.

Al momento in cui il Foundation Level Agile Extension Syllabus è stato completato, il gruppo di lavoro Agile Extension era costituto dai seguenti membri:

Rex Black (Chair), Bertrand Cornanguer (Vice Chair), Gerry Coleman (Learning Objectives Lead), Debra Friedenberg (Exam Lead), Alon Linetzki (Business Outcomes e Marketing Lead), Tauhida Parveen (Editor) e Leo van der Aalst (Development Lead)..

Autori: Rex Black, Anders Claesson, Gerry Coleman, Bertrand Cornanguer, Istvan Forgacs, Alon Linetzki, Tilo Linz, Leo van der Aalst, Marie Walsh e Stephan Weber

Revisori interni: Mette Bruhn-Pedersen, Christopher Clements, Alessandro Collino, Debra Friedenberg, Kari Kakkonen, Beata Karpinska, Sammy Kolluru, Jennifer Leger, Thomas Mueller, Tuula Pääkkönen, Meile Posthuma, Gabor Puhalla, Lloyd Roden, Marko Rytkönen, Monika Stoecklein-Olsen, Robert Treffny, Chris Van Bael e Erik van Veenendaal

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

Questo documento è stato formalmente rilasciato dall'Assemblea Generale dell'ISTQB® il 31 maggio 2014.

Traduzione italiana: si ringraziano Massimo Di Carlo (Rev. 1 - 2014) e Cristina Sobrero (Rev. 2 - 2022)

0. Introduzione

0.1 Scopo di questo Documento

Questo Syllabus è la base per l'International Software Testing Qualification di Livello Foundation per Agile Tester.

L'ISTQB® mette a disposizione questo syllabus con le seguenti modalità:

1. Ai Membri del Board, per consentirne la traduzione nella loro lingua madre e di accreditare i fornitori della formazione. I Board nazionali possono adattare il syllabus alle loro particolari esigenze linguistiche e modificare i riferimenti bibliografici per adattarli alle loro pubblicazioni locali.
2. Agli Enti Esaminatori, per consentire la traduzione delle domande d'esame nella loro lingua adattandole agli obiettivi di apprendimento di ogni modulo.
3. Ai Fornitori della formazione, per consentire la produzione di materiale didattico e mettere a punto metodologie d'insegnamento appropriate.
4. Ai Candidati alla certificazione, per consentire la preparazione all'esame (partecipando a un corso o in modo indipendente).
5. Alla Comunità che si occupa d'ingegneria dei sistemi e del software, per consentire di migliorare la professione di tester del software e dei sistemi e per fornire una base di riferimento per libri ed articoli.

L'ISTQB® permette inoltre ad altre entità di usare questo syllabus per altre finalità, purché ne chiedano ed ottengano anticipatamente un permesso scritto.

0.2 Panoramica

Il documento Overview dell'estensione Agile al Foundation Level[ISTQB_AL_OVIEW] include le seguenti informazioni:

- Risultati di business per il Syllabus
- Sommario del Syllabus
- Relazioni fra i Syllabi
- Descrizione dei Livelli di Conoscenza (Livelli K)
- Appendici

0.3 Obiettivi di Apprendimento Esaminabili

Gli obiettivi di apprendimento supportano i risultati di business e sono utilizzati per progettare gli esami volti ad ottenere la certificazione a livello Foundation – Agile Tester. In generale tutte le sezioni di questo Syllabus sono esaminabili ad un livello K1. Cioè il candidato dovrà riconoscere, ricordare e richiamare un termine o un concetto. Gli obiettivi di apprendimento per i livelli K1, K2 e K3 sono mostrati all'inizio di ogni corrispondente capitolo.

1. Sviluppo Software Agile – 150 minuti

Parole Chiave:

Base di test, ciclo di vita dello sviluppo software, Manifesto Agile, modello di sviluppo incrementale, modello di sviluppo iterativo, oracolo del test, sviluppo software Agile, test automation, Test-Driven development, user story

Obiettivi di Apprendimento per lo Sviluppo Software Agile

1.1 I fondamenti dello Sviluppo Software Agile

FA-1.1.1 (K1) Ricordare i concetti base dello sviluppo software Agile descritti nel Manifesto Agile

FA-1.1.2 (K2) Capire i vantaggi dell'approccio whole-team

FA-1.1.3 (K2) Capire i benefici di feedback anticipati e frequenti

1.2 Aspetti degli approcci Agili

FA-1.2.1 (K1) Ricordare gli approcci dello sviluppo software Agile

FA-1.2.2 (K3) Scrivere user story testabili in collaborazione con gli sviluppatori e i rappresentanti di business.

FA-1.2.3 (K2) Capire come le retrospettive possono essere usate come meccanismo per migliorare il processo.

FA-1.2.4 (K2) Capire l'uso e l'obiettivo del continuous integration.

FA-1.2.5 (K1) Conoscere le differenze tra pianificazione della release e pianificazione dell'iterazione, e come un tester aggiunge valore in ciascuna di queste attività.

1.1 I Fondamenti dello Sviluppo Software Agile

Un tester in un progetto Agile lavora in modo differente rispetto a un tester che lavora su un progetto tradizionale. I tester devono comprendere i valori e i principi che sono alla base dei progetti Agile, e come i tester sono parte integrante di un approccio di squadra insieme agli sviluppatori e ai rappresentanti di business. I membri di un progetto Agile comunicano tra loro fin dalle prime fasi del progetto e frequentemente, e questo aiuta a eliminare i difetti nelle fasi iniziali e a sviluppare un prodotto di qualità.

1.1.1 Sviluppo Software Agile e Manifesto Agile

Nel 2001, un gruppo di persone, che rappresentavano le più diffuse metodologie di sviluppo software leggere (leightweight), ha concordato un insieme comune di valori e di principi che è stato chiamato Manifesto for Agile Software Development o Manifesto Agile [Agilemanifesto]. Il Manifesto Agile contiene quattro dichiarazioni di valori:

- Gli individui e le interazioni più che i processi e gli strumenti
- Il software funzionante più che la documentazione esaustiva
- La collaborazione col cliente più che la negoziazione dei contratti

- Rispondere al cambiamento più che seguire un piano

Il Manifesto Agile afferma che, sebbene i concetti sulla destra abbiano valore, i concetti a sinistra hanno un valore maggiore.

Individui e Interazioni

Lo sviluppo Agile è molto incentrato sulle persone. Team di persone costruiscono il software ed è attraverso un'interazione e una comunicazione continua, piuttosto che facendo affidamento a strumenti o processi, che i team possono lavorare in modo più efficace.

Software Funzionante

Dal punto di vista del cliente, un software funzionante è molto più utile e di valore rispetto a una documentazione eccessivamente dettagliata, e inoltre offre l'opportunità di dare al team di sviluppo un feedback rapido. Infine, poiché un software funzionante, anche se con funzionalità ridotte, è disponibile molto prima nel ciclo di sviluppo, lo sviluppo Agile può portare a un vantaggio significativo sul time-to-market. Lo sviluppo Agile è quindi particolarmente utile in ambienti di business in rapida evoluzione in cui i problemi e/o le soluzioni non sono chiare o in cui il business desidera innovarsi verso nuovi ambiti.

Collaborazione con il Cliente

I clienti spesso incontrano molta difficoltà a specificare il sistema di cui hanno bisogno. Collaborando direttamente con il cliente aumenta la probabilità di comprendere esattamente quello che il cliente richiede. Avere contratti con i clienti può essere importante, ma lavorare in collaborazione stretta e regolare con i clienti aumenta la probabilità di raggiungere un maggiore successo del progetto.

Rispondere al Cambiamento

Il cambiamento è inevitabile nei progetti software. L'ambiente in cui il business opera, gli aspetti legislativi, l'attività dei competitor, i progressi della tecnologia e altri fattori possono avere impatti importanti sul progetto e i suoi obiettivi. Questi fattori devono essere adeguati al processo di sviluppo. In tal caso, la flessibilità delle pratiche di lavoro per adattarsi ai cambiamenti è più importante che semplicemente aderire in modo rigido a un piano.

Principi

- I valori chiave del Manifesto Agile sono catturati in 12 principi:
- La nostra massima priorità è soddisfare il cliente rilasciando software di valore, il prima possibile e in maniera continuativa
- Accogliamo i cambiamenti nei requisiti, anche nelle fasi avanzate dello sviluppo . I processi Agile sfruttano il cambiamento a favore del vantaggio competitivo del cliente
- Rilasciare frequentemente software funzionante, con cadenza variabile da un paio di settimane a un paio di mesi, preferendo i periodi brevi
- Le persone del business e gli sviluppatori devono lavorare insieme quotidianamente per tutta la durata del progetto
- Fondiamo i progetti su individui motivati. Diamo loro l'ambiente e il supporto di cui hanno bisogno e confidiamo nella loro capacità di portare il lavoro a termine.
- Una conversazione face-to-face è il metodo più efficiente ed efficace di trasmettere informazioni con il team e all'interno del team di sviluppo
- Il software funzionante è la principale misura del progresso
- I processi Agile promuovono lo sviluppo sostenibile. Gli sponsor, gli sviluppatori e gli utenti dovrebbero essere in grado di mantenere indefinitamente un ritmo costante.
- La continua attenzione all'eccellenza tecnica e alla buona progettazione esaltano l'agilità.
- La semplicità - l'arte di massimizzare la quantità di lavoro non svolto – è fondamentale.
- Le architetture, i requisiti e le progettazioni migliori emergono da team che si auto-organizzano.

- A intervalli regolari, il team riflette su come diventare più efficace, dopodichè regola e adatta il proprio comportamento di conseguenza.

Le diverse metodologie Agili forniscono procedure pratiche per mettere in atto questi valori e principi.

1.1.2 Approccio Whole-Team

L'approccio whole-team significa coinvolgere tutti quelli che hanno le conoscenze e le competenze necessarie per garantire il successo del progetto. Il team comprende rappresentanti del cliente e altri stakeholder di business che determinano le caratteristiche del prodotto. Il team dovrebbe essere relativamente piccolo; sono stati osservati team di successo con un minimo di tre persone fino a nove persone. Idealmente, tutto il team condivide lo stesso spazio di lavoro, perché la co-locazione agevola fortemente la comunicazione e l'interazione. L'approccio whole-team è gestito attraverso i daily stand-up meeting (si veda Paragrafo 2.2.1) che coinvolgono tutti i membri del team, dove viene comunicato l'avanzamento delle attività (task) e vengono evidenziati eventuali impedimenti all'avanzamento. L'approccio whole-team promuove dinamiche di gruppo più efficaci ed efficienti.

L'uso dell'approccio whole-team per lo sviluppo è uno dei principali benefici dello sviluppo Agile. I suoi benefici includono:

- Migliorare la comunicazione e la collaborazione all'interno del team
- Consentire di utilizzare le diverse competenze all'interno del team a vantaggio del progetto
- Rendere la qualità una responsabilità di tutti

L'intero team è responsabile della qualità nei progetti Agile. L'essenza dell'approccio whole team sta nel fatto che i tester, gli sviluppatori e i rappresentanti del business lavorano insieme in ogni fase del processo di sviluppo. I tester lavoreranno a stretto contatto con gli sviluppatori e i rappresentanti di business per garantire che i livelli di qualità desiderati siano raggiunti. Questo include il supporto e la collaborazione con i rappresentanti di business per aiutarli a creare test di accettazione adeguati, la collaborazione con gli sviluppatori per concordare la strategia di test e decidere l'approccio al test automation. I tester possono così trasferire ed estendere la conoscenza del testing agli altri membri del team e influenzare lo sviluppo del prodotto.

L'intero team è coinvolto in tutte le discussioni o i meeting in cui sono presentate, analizzate o stimate le funzionalità del prodotto. Il concetto di coinvolgere i tester, gli sviluppatori e i rappresentanti di business in tutte le discussioni sulle funzionalità è conosciuto come "Potere dei Tre" (Power of Three) [Crispin08].

1.1.3 Feedback Anticipati e Frequenti

I progetti Agile hanno iterazioni brevi che consentono al team di progetto di ricevere feedback anticipati e continui sulla qualità del prodotto, durante tutto il ciclo di sviluppo. Un modo per fornire feedback rapido è grazie al Continuous Integration (si veda paragrafo 1.2.4).

Quando si utilizzano approcci di sviluppo sequenziali, il cliente spesso non vede il prodotto fino a quando il progetto è quasi completato. A quel punto è spesso troppo tardi per il team di sviluppo indirizzare in modo efficace eventuali problemi che il cliente possa evidenziare. Ottenendo frequenti feedback dai clienti mentre il progetto avanza, i team Agile possono integrare la maggior parte delle nuove modifiche nel processo di sviluppo del prodotto. Feedback frequenti e anticipati aiutano il team a focalizzarsi sulle funzionalità con il più alto valore di business o rischio associato, che verranno rilasciate prima al cliente. Aiuta anche a gestire meglio il team in quanto la sua capacità è trasparente per tutti. Ad esempio, quanto lavoro può essere svolto in uno sprint o iterazione? Che cosa potrebbe aiutarci ad essere più veloci? Che cosa ci impedisce di farlo?

I benefici di feedback anticipati e frequenti includono:

- Evitare incomprensioni sui requisiti, che possono non essere rilevati fino alle ultime fasi nel ciclo di sviluppo, quando sono più costosi da correggere.
- Comprendere le richieste dei clienti, rendendole disponibili al cliente per l'utilizzo in anticipo. In questo modo, il prodotto corrisponderà meglio a quello il cliente desidera.
- Rilevare (attraverso il Continuous Integration), isolare e risolvere in anticipo i problemi di qualità.
- Fornire informazioni al team Agile sulla sua produttività e capacità di rilascio.
- Promuovere un slancio consistente al progetto.

1.2 Aspetti degli Approcci Agile

Esistono un certo numero di approcci Agile utilizzati dalle organizzazioni. Pratiche comuni tra le organizzazioni Agile includono la creazione collaborativa delle user story, la retrospettiva, il Continuous Integration e la pianificazione di ogni iterazione, oltre al rilascio completo. Questo paragrafo descrive alcuni degli approcci Agile.

1.2.1 Approcci allo Sviluppo Software Agile

Esistono diversi approcci Agile, ognuno dei quali implementa i valori e i principi del Manifesto Agile in modo differente. In questo Syllabus sono stati considerati tre approcci Agile rappresentativi: Extreme Programming (XP), Scrum e Kanban.

Extreme Programming

Extreme Programming (XP), originariamente introdotto da Kent Beck [Beck04], è un approccio Agile allo sviluppo software descritto da specifici valori, principi e pratiche di sviluppo.

XP abbraccia cinque valori per guidare lo sviluppo: comunicazione, semplicità, feedback, coraggio e rispetto.

XP descrive un insieme di principi come linee guida aggiuntive: umanità, economia, mutuo vantaggio, auto-similarità, miglioramento, diversità, riflessione, flusso, opportunità, ridondanza, fallimento, qualità, piccoli passi e responsabilità accettata.

XP prescrive tredici pratiche principali: sedersi insieme (sit together), whole-team, area di lavoro informativa (informative workspace), lavoro motivato (energized work o sustainable Pace), pair programming, user story, ciclo settimanale (weekly cycle), ciclo trimestrale (quarterly cycle), slack, ten–minutes build, Continuous Integration, test-first programming e progettazione incrementale.

Molti degli approcci di sviluppo software Agile ad oggi in uso sono influenzati da XP e dai suoi valori e principi. Ad esempio, i team Agile che seguono Scrum incorporano spesso pratiche XP.

Scrum

Scrum è un framework di management dello sviluppo Agile che include i seguenti strumenti e le seguenti pratiche [Schwaber01]:

- Sprint: Scrum divide un progetto in iterazioni (chiamate sprint) di lunghezza fissa (di solito 2-4 settimane).
- Incremento di Prodotto (Product Increment): Ogni sprint crea un prodotto potenzialmente rilasciabile / vendibile (chiamato incremento).
- Product Backlog: Il Product Owner gestisce una lista prioritizzata di elementi di prodotto pianificati (chiamato Product Backlog). Il Product Backlog evolve ad ogni sprint (chiamato raffinamento del backlog, baklog refinement).
- Sprint Backlog: all'inizio di ogni sprint, il team Scrum seleziona un insieme di elementi con la priorità più alta (chiamato Sprint Backlog) dal Product Backlog. Poiché il team Scrum, non il

Product Owner, seleziona gli elementi da realizzare durante lo sprint, la selezione è si basa sul principio pull piuttosto che sul principio push.

- Definition of Done: per assicurarsi che esista un prodotto potenzialmente rilasciabile alla fine di ogni sprint, il team Scrum discute e definisce i criteri appropriati per il completamento dello sprint. La discussione approfondisce la comprensione da parte del team degli elementi del backlog e dei requisiti di prodotto.
- Timeboxing: Solo le attività (task), i requisiti o le funzionalità che il team prevede di terminare durante lo sprint sono parte dello Sprint Backlog. Se il team di sviluppo non può finire un task all'interno di uno sprint, le funzionalità associate vengono eliminate dallo sprint e il task viene spostato nel Product Backlog. Il Timeboxing si applica non solo ai task, ma anche in altre situazioni (ad esempio, far rispettare l'inizio e la fine di un meeting).
- Trasparenza: Il team di sviluppo fornisce quotidianamente una sintesi sullo stato dello sprint e lo aggiorna durante il daily scrum meeting. Questo rende visibile il contenuto e l'avanzamento dello sprint attuale, compresi i risultati dei test, al team, al management e a tutte le parti interessate. Ad esempio, il team di sviluppo può visualizzare lo stato dello sprint su una whiteboard (lavagna bianca).

Scrum definisce tre ruoli:

- Scrum Master: assicura che le pratiche e le regole Scrum siano implementate e rispettate, e risolve ogni violazione, problematica sulle risorse o altro impedimento che potrebbe impedire al team di seguire le pratiche e le regole. Questa persona non è il team leader, ma un coach.
- Product Owner: rappresenta il cliente e genera, mantiene e prioritizza il Product Backlog. Questa persona non è il team leader.
- Team di sviluppo: sviluppa e testa il prodotto. Il team è auto-organizzato: Non esiste un team leader, quindi il team prende le decisioni. Il team è anche cross-funzionale (si vedano i paragrafi 2.3.2 e 3.1.4).

Scrum (rispetto a XP) non detta specifiche tecniche di sviluppo software (per es. test-first programming). In aggiunta, Scrum non fornisce una guida su come deve essere fatto il testing in un progetto Scrum.

Kanban

Kanban è un approccio di management viene usato alcune volte nei progetti Agile. L'obiettivo generale è visualizzare e ottimizzare il flusso di lavoro all'interno di una value-added chain (catena del valore aggiunto). Kanban utilizza tre strumenti [Linz14]:

- Kanban Board: La value chain da gestire è visualizzata in una lavagna Kanban Board. Ogni colonna visualizza una stazione, che è un insieme di task specifici, ad esempio i task di sviluppo o i task di test. Gli elementi che devono essere prodotti o i task che devono essere eseguiti sono rappresentati da ticket che si spostano da sinistra verso destra sulla Kanban Board, attraverso le stazioni.
- Work-in-Progress Limit: il numero di task paralleli attivi è strettamente limitato. Questo è controllato dal massimo numero di ticket permessi in una stazione e/o globalmente in una Kanban Board. Ogni volta che una stazione ha capacità disponibile, viene preso un ticket dalla precedente stazione.
- Lead Time: Kanban è usato per ottimizzare il flusso continuo di task minimizzando il tempo (medio) di un task dalla sua entrata alla sua uscita (lead time) nel value stream completo.

Kanban è simile a Scrum. In entrambi i framework, visualizzare i task attivi (ad es. su una whiteboard visibile a tutti) fornisce trasparenza del contenuto e dell'avanzamento dei task. I task non ancora schedulati rimangono in un backlog e vengono spostati sulla Kanban board non appena esiste spazio (capacità produttiva) disponibile.

Iterazioni o sprint sono opzionali in Kanban. Il processo Kanban consente di rilasciare elemento per elemento, piuttosto che come parte di una release. Il Timeboxing come meccanismo di sincronizzazione è quindi opzionale, mentre il timeboxing in Scrum sincronizza tutti i task all'interno di uno sprint.

1.2.2 Creazione Collaborativa delle User Story

Specifiche di scarsa qualità sono spesso una delle principali ragioni per il fallimento del progetto. Problemi nelle specifiche possono derivare dalla mancanza da parte degli utenti di comprensione dei loro veri bisogni, assenza di una visione globale del sistema, funzionalità ridondanti o contraddittorie e altre difficoltà di comunicazione. Nello sviluppo Agile, le user story sono scritte per catturare i requisiti dal punto di vista degli sviluppatori, tester e rappresentanti di business. Nello sviluppo sequenziale, questa visione condivisa di una funzionalità si realizza attraverso review formali dopo che sono stati scritti i requisiti; nello sviluppo Agile, questa visione condivisa si realizza attraverso frequenti review informali mentre vengono scritti i requisiti.

Le user story devono indirizzare sia le caratteristiche funzionali che non-funzionali. Ogni user story include i criteri di accettazione per queste caratteristiche. Tali criteri dovrebbero essere definiti attraverso una collaborazione tra i rappresentanti di business, gli sviluppatori e i tester. I criteri forniscono agli sviluppatori e ai tester una visione estesa della funzione che i rappresentanti di business valideranno. Un team Agile considera terminato un task quando un insieme di criteri di accettazione è stato soddisfatto.

Tipicamente, la prospettiva del tester migliorerà la user story identificando dettagli mancanti o requisiti non-funzionali. Un tester può contribuire ponendo ai rappresentanti di business domande aperte sulla user story, proponendo modi per testarla e confermando i criteri di accettazione

La scrittura collaborativa della user story può utilizzare tecniche come il brainstorming e le mappe mentali. Il tester può utilizzare la tecnica INVEST [INVEST]:

- **I**ndipendente (independent)
- **N**egoziabile (Negotiable)
- **V**alutabile (Valuable)
- Stimabile (***E****stimable*)
- Piccola (***S****mall*)
- **T**estabile (Testable)

Secondo il concetto 3C [Jeffries00], una user story è la combinazione di tre elementi:

- Card: La card è il dispositivo fisico che descrive una user story. Identifica il requisito, la sua criticità, la durata prevista per lo sviluppo e il test, e i criteri di accettazione per quella user story. La descrizione deve essere accurata, poiché verrà utilizzata nel product backlog.
- Conversazione (Conversation): La conversazione spiega come verrà utilizzato il software. La conversazione può essere documentata o verbale. I tester, avendo un punto di vista diverso rispetto agli sviluppatori e ai rappresentanti di business [ISTQB_FL_SYL], forniscono un valido contributo allo scambio di idee, opinioni ed esperienze. La conversazione inizia durante la fase di pianificazione della release (release-planning) e continua fino a quando viene schedulata la user story.
- Conferma (Confirmation): I criteri di accettazione, discussi nella conversazione, vengono utilizzati per confermare che la user story è completata. Questi criteri di accettazione possono estendersi su più user story. Dovrebbero essere utilizzati sia test positivi che negativi per coprire i criteri. Durante la conferma, diversi partecipanti svolgono il ruolo di tester. Questi possono includere sviluppatori o specialisti nell'ambito delle prestazioni, sicurezza, interoperabilità e altre caratteristiche di qualità. Per confermare una user story come completata, i criteri di

accettazione definiti dovrebbero essere testati e validati per dimostrare che sono stati soddisfatti.

I team Agile possono documentare in modo differente le user story. Indipendentemente dall'approccio adottato, la documentazione delle user story dovrebbe essere concisa, sufficiente e necessaria.

1.2.3 Retrospettive

Nello sviluppo Agile, una retrospettiva è un meeting svolto al termine di ogni iterazione per discutere su cosa ha avuto successo, cosa potrebbe essere migliorato e come incorporare i miglioramenti e mantenere i successi nelle iterazioni future. Le retrospettive coprono argomenti come il processo, le persone, le organizzazioni, le relazioni e gli strumenti. Retrospettive svolte regolarmente, quando si verificano attività di follow-up appropriate, sono fondamentali per l'auto organizzazione e il miglioramento continuo dello sviluppo e del testing.

Le retrospettive possono portare a decisioni di miglioramento legate al testing, focalizzate sull'efficacia e la produttività del testing, sulla qualità dei test case e sulla soddisfazione del team. Possono anche indirizzare la testabilità delle applicazioni, delle user story, delle funzioni o delle interfacce di sistema. La root cause analysis dei difetti può guidare il testing o migliorare lo sviluppo. In generale, i team dovrebbero implementare solo pochi miglioramenti per iterazione. Questo consente un miglioramento continuo in modo sostenibile.

Le tempistiche e l'organizzazione della retrospettiva dipendono dalla particolare metodologia Agile seguita. I rappresentanti di business e il team partecipano ad ogni retrospettiva come partecipanti, mentre il facilitatore organizza ed esegue il meeting. In alcuni casi, i teams possono invitare altri partecipanti al meeting.

I tester dovrebbero svolgere un ruolo importante nelle retrospettive. I tester sono parte del team e portano il loro punto di vista unico ([ISTQB_FL_SYL], Par. 1.5). Il testing viene eseguito in ogni sprint e contribuisce in modo determinante al successo. Tutti i membri del team, tester e non-tester, possono fornire un contributo sia alle attività di test che alle attività non di test.

Le retrospettive devono avvenire all'interno di un ambiente professionale caratterizzato da fiducia reciproca. Gli attributi di una retrospettiva di successo sono gli stessi di quelli di ogni altra review, come riportati in [ISTQB_FL_SYL] (Par. 3.2).

1.2.4 Continuous Integration

Il rilascio di un Incremento di Prodotto richiede un software integrato, funzionante e affidabile alla fine di ogni sprint. Il Continuous Integration indirizza questa sfida inserendo tutte le modifiche al software e integrando tutti i componenti modificati in modo regolare, almeno una volta al giorno. Il Configuration Management, la compilazione, la build del software, l'installazione e il testing sono inseriti in un unico processo automatizzato e ripetibile. Poiché gli sviluppatori integrano il loro lavoro, effettuano la build ed eseguono il testing in modo regolare e continuo, i difetti nel codice sono rilevati più velocemente.

Quando gli sviluppatori hanno eseguito la codifica, il debugging e il check-in del codice in un repository condiviso di codice sorgente, un processo di Continuous Integration esegue le seguenti attività automatizzate:

- Analisi statica del codice: esecuzione dell'analisi statica del codice e reporting dei risultati
- Compilazione: compilazione e linking del codice, generando i file eseguibili
- Unit test: esecuzione degli unit test, verificando la copertura del codice, e reporting dei risultati
- Deploy: installazione della build nell'ambiente di test
- Testing di integrazione: esecuzione dei test di integrazione e reporting dei risultati
- Reporting (dashboard): pubblicazione dello stato di tutte queste attività in un luogo visibile e condiviso, oppure attraverso e-mail al team.

Un processo di build e test automatizzato viene svolto su base giornaliera e rileva errori di integrazione in anticipo e velocemente. Il Continuous Integration permette ai tester Agile di eseguire test automatizzati regolarmente, in alcuni casi come parte dello stesso processo di Continuous Integration, inviando al team un rapido feedback sulla qualità del codice. Questi risultati dei test sono visibili a tutti i membri del team, soprattutto quando report automatizzati sono integrati nel processo. Il regression testing automatizzato può essere eseguito in modo continuativo durante l'iterazione. Regression test validi coprono il maggior numero di funzionalità possibile, incluse le user story rilasciate nelle precedenti iterazioni. Una buona copertura nei regression test automatizzati aiuta a supportare la build (e il testing) di grandi sistemi integrati. Quando il regression testing è automatizzato, i tester Agile sono liberi di concentrarsi sul testing manuale delle nuove funzionalità e delle modifiche implementate e sul testing confermativi delle correzioni dei difetti.

In aggiunta ai test automatizzati, le organizzazioni che utilizzano il Continuous Integration, tipicamente usano strumenti di build per implementare un continuo Quality Control. Oltre ad eseguire unit test e test di integrazione, tali strumenti possono eseguire test aggiuntivi statici e dinamici, misurare e profilare le prestazioni, estrarre e formattare la documentazione dal codice sorgente, e facilitare i processi manuali di quality assurance. Questa continua applicazione del quality control ha lo scopo di migliorare la qualità del prodotto e di ridurre il tempo impiegato per rilasciare il prodotto stesso, sostituendo la pratica tradizionale di applicare il quality control dopo aver completato tutto lo sviluppo.

Gli strumenti di build possono essere integrati con strumenti per l'installazione automatica, che possono estrarre la build corretta dal server di Continuous Integration o dal server di build, e installarla in uno o più ambienti di sviluppo, di test, di staging o anche in produzione. Questo riduce gli errori e i ritardi associati all'affidarsi a staff specializzato o a programmatori per installare le release in tali ambienti.

Il Continuous Integration può dare i seguenti benefici:

- Permette di rilevare in anticipo ed eseguire più facilmente la root cause analysis di problemi nell'integrazione e in modifiche inconsistenti tra loro
- Fornisce al team di sviluppo un feedback regolare sulla correttezza del codice
- Mantiene per un giorno la versione software sviluppata che deve essere testata
- Riduce il rischio di regressione associato al refactoring del codice grazie alla veloce ripetizione del testing del codice dopo ogni piccolo insieme di modifiche
- Fornisce confidenza che ogni sviluppo implementato quotidianamente sia basato su solide fondamenta
- Rende visibili i progressi rispetto al completamento dell'Incremento di Prodotto, incoraggiando gli sviluppatori e i tester
- Elimina i rischi di schedulazione associati all'integrazione big-bang
- Fornisce durante tutto lo sprint una disponibilità continua di software eseguibile per il testing, le demo o per scopi formativi
- Riduce le attività ripetitive del testing manuale
- Fornisce un rapido feedback sulle decisioni prese per migliorare la qualità e i test

Comunque, il Continuous Integration non è privo di rischi e sfide:

- Devono essere introdotti e manutenuti strumenti di Continuous Integration
- Il processo di Continuous Integration deve essere definito e stabilito
- Il test automation richiede risorse aggiuntive e può essere complesso da avviare
- Una copertura totale dei test è fondamentale per ottenere i vantaggi del testing automatizzato
- I team a volte fanno un eccessivo affidamento sugli unit test ed eseguono troppo pochi test di sistema e di accettazione

Il Continuous Integration richiede l'uso di strumenti, inclusi strumenti per il testing, strumenti per automatizzare il processo di build e strumenti per il controllo delle versioni.

1.2.5 Pianificazione della Release e Pianificazione dell'Iterazione

Come riportato in [ISTQB_FL_SYL], la pianificazione è un'attività continua e questo è vero anche nello sviluppo software Agile. Nello sviluppo software Agile, esistono due tipi di pianificazione: la pianificazione della release (release planning) e la pianificazione dell'iterazione (iteration planning).

La pianificazione della release prevede il rilascio di un prodotto, spesso un paio di mesi prima dell'inizio di un progetto. La pianificazione della release definisce e ri-definisce il Product Backlog e può comportare il raffinamento delle user story più grandi in un insieme di user story più piccole. La pianificazione della release fornisce la base per un approccio del test e un Test Plan che copre tutte le iterazioni. I release plan sono di alto livello.

Nella pianificazione della release, i rappresentanti di business stabiliscono e prioritizzano le user story per la release, in collaborazione con il team (si veda par.1.2.2). Sulla base di queste user story, vengono identificati i rischi di progetto e i rischi di qualità, e viene eseguita una stima dell'effort di alto livello (si veda par.3.2).

I tester sono coinvolti nella pianificazione della release e aggiungono valore specialmente nelle seguenti attività:

- Definizione di user story testabili, che includano i criteri di accettazione
- Partecipazione all'analisi dei rischi di progetto e dei rischi di qualità
- Stima dell'effort per il testing associato alle user story
- Definizione dei livelli di test necessari
- Pianificazione del testing per la release

Dopo aver eseguito la pianificazione della release, inizia la pianificazione dell'iterazione per la prima iterazione. La pianificazione dell'iterazione prevede le attività fino alla fine di una singola iterazione e riguarda lo Sprint Backlog.

Nella pianificazione dell'iterazione, il team seleziona le user story dal release backlog prioritizzato, elabora le user story, esegue un'analisi del rischio per le user story, e stima il lavoro necessario per ogni user story. Se una user story è troppo vaga e i tentativi di chiarirla sono falliti, il team può rifiutarsi di accettarla e considera la user story successiva in ordine di priorità. I rappresentanti di business devono rispondere alle domande del team sulla user story, in modo che il team possa comprendere meglio quello che dovrebbe implementare e come dovrebbe testare ogni user story.

Il numero di user story selezionate si basa sulla velocità del team e la dimensione stimata delle user story selezionate. Dopo che sono stati finalizzati i contenuti dell'iterazione, le user story vengono suddivise in task, che saranno eseguiti dai membri del team appropriati.

I tester sono coinvolti nella pianificazione dell'iterazione e aggiungono valore nelle seguenti attività:

- Partecipazione nell'analisi dettagliata del rischio delle user story
- Valutazione della testabilità delle user story
- Creazione dei test di accettazione per le user story
- Suddivisione delle user story in task (in particolare task di test)
- Stima dell'effort del testing per tutti i task di test
- Identificazione degli aspetti funzionali e non-funzionali del sistema da testare
- Supporto e partecipazione al test automation nei vari livelli di test.

I release plan possono modificarsi con l'avanzamento del progetto, incluse le modifiche a singole user story nel Product Backlog. Questi cambiamenti possono essere attivati da fattori interni o esterni. I fattori interni includono la capacità di rilascio, la velocità e problemi tecnici. I fattori esterni includono la scoperta di nuovi mercati e opportunità, nuovi competitor o problemi di business che possono modificare gli obiettivi e/o le date della release. Inoltre, gli iteration plan possono modificarsi durante l'iterazione. Ad esempio, una particolare user story che è stata considerata relativamente semplice durante la stima potrebbe rivelarsi più complessa del previsto.

Queste modifiche possono essere sfidanti per i tester. I tester devono comprendere il quadro generale della release ai fini della pianificazione dei test, e devono avere disponibile una base di test e un oracolo del test adeguati in ogni iterazione per gli scopi del processo di test, come descritto in [ISTQB_FL_SYL] (Par.1.4). Le informazioni richieste devono essere disponibili al tester in anticipo, e anche la modifica deve essere gestita secondo i principi Agile. Questa dilemma richiede decisioni attente sulle strategie di test e sulla documentazione di test. Per ulteriori informazioni sulle sfide del testing Agile, si veda [Black09] (Cap.12).

La pianificazione della release e la pianificazione dell'iterazione devono indirizzare la pianificazione dei test e la pianificazione delle attività di sviluppo. Problemi specifici del test includono:

- L'ambito del testing, l'estensione del testing per quelle aree in ambito, gli obiettivi del test e le motivazioni di queste decisioni.
- I membri del team che eseguiranno le attività di test.
- L'ambiente di test e i dati di test, se necessari, e qualsiasi inserimento o modifica all'ambiente e/o ai dati di test che può accadere prima o durante il progetto.
- La tempistica, la sequenza, le dipendenze e i prerequisiti per le attività di test funzionali e non-funzionali (ad es. la frequenza di esecuzione dei regression test, le funzionalità che dipendono da altre funzionalità, i dati di test, ecc.), includendo come le attività di test sono correlate e dipendono dalle attività di sviluppo.
- I rischi di progetto e i rischi di qualità che devono essere indirizzati (si veda pr.3.2.1).

In aggiunta, il più ampio sforzo di stima del team dovrebbe prendere in considerazione il tempo e l'effort necessari per completare le attività di test richieste.

2. Principi, Regole e Processi Fondamentali del Testing Agile – 105 minuti

Parole Chiave:

Build verification test, configuration item, configuration management.

Obiettivi di Apprendimento per Principi, Regole e Processi Fondamentali del Testing Agile.

2.1 Le Differenze tra il Testing nell'Approccio Tradizionale e nell'Approccio Agile

FA-2.1.1 (K2) Descrivere le differenze tra le attività di test nei progetti Agile e progetti non-Agile

FA-2.1.2 (K2) Descrivere come le attività di sviluppo e di test sono integrate nei progetti Agile.

FA-2.1.3 (K2) Descrivere il ruolo del testing indipendente nei progetti Agile.

2.2 Stato del Testing nei Progetti Agile

FA-2.2.1 (K2) Descrivere gli strumenti e le tecniche usati per comunicare lo stato del testing in un progetto Agile, inclusi i progressi del testing e la qualità del prodotto

FA-2.2.2 (K2) Descrivere il processo di evoluzione dei test attraverso iterazioni multiple, e spiegare perché il test automation è importante per gestire il rischio di regressioni nei progetti Agile.

2.2 Ruolo e Competenze di un Tester in un Team Agile

FA-2.3.1 (K2) Comprendere le competenze (persona, dominio di conoscenza e testing) di un tester in un team Agile

FA-2.3.2 (K2) Comprendere il ruolo di un tester in un team Agile.

2.1 Le Differenze tra il Testing nell'Approccio Tradizionale e nell'Approccio Agile

Come descritto in [ISTQB_FL_SYL] e in [Black09], le attività di test sono correlate alle attività di sviluppo, e quindi il testing varia in base ai differenti cicli di vita dello sviluppo software. I tester devono comprendere le differenze tra il testing nei modelli di ciclo di vita tradizionali (ad esempio, sequenziali come il V-model o iterativi come RUP) e nei modelli di ciclo di vita Agile, in modo da poter lavorare in modo efficace ed efficiente. I modelli Agile differiscono in base a come vengono integrate le attività di test e di sviluppo, in base ai prodotti di lavoro del progetto, ai termini, ai criteri di ingresso e di uscita per i vari livelli di test, all'uso di strumenti e a come il testing indipendente può essere utilizzato in modo efficace.

I tester dovrebbero ricordare che le organizzazioni variano notevolmente nella loro implementazione del ciclo di vita dello sviluppo software. Una deviazione dagli ideali del ciclo di vita Agile (si veda par.1.1) può rappresentare una personalizzazione e un adattamento intelligente delle pratiche. La capacità di adattarsi al contesto di un determinato progetto, comprese le pratiche di sviluppo software effettivamente seguite, è un fattore chiave di successo per i tester.

2.1.1 Attività di Test e di Sviluppo

Una delle principali differenze tra i cicli di vita tradizionali e cicli di vita Agile è il concetto di iterazioni molto brevi, dove ogni iterazione rilascia software funzionante che implementa funzionalità di valore per gli stakeholder di business. All'inizio del progetto, esiste un periodo di pianificazione della release. Questo è seguito da una sequenza di iterazioni. All'inizio di ogni iterazione, esiste un periodo di pianificazione dell'iterazione. Una volta stabiliti l'ambito dell'iterazione, le user story selezionale vengono implementate, integrate nel sistema e testate. Queste iterazioni sono altamente dinamiche, e le attività di sviluppo, integrazione e test si svolgono durante ogni iterazione con notevole parallelismo e sovrapposizioni. Le attività di test vengono svolte durante l'iterazione e non come attività finale.

Tester, sviluppatori e stakeholder di business hanno tutti un ruolo durante il testing, come nei cicli di vita tradizionali. Gli sviluppatori eseguono gli unit test dopo l'implementazione delle funzionalità delle user story. I tester eseguono successivamente il testing di queste funzionalità. Anche gli stakeholder di business eseguono il testing delle user story durante l'implementazione. Gli stakeholder di business potrebbero utilizzare test case scritti, ma potrebbero semplicemente eseguire un test esplorativo e utilizzare la funzionalità per fornire un feedback veloce al team di sviluppo.

In alcuni casi, si svolgono periodicamente iterazioni di consolidamento (hardening), o di stabilizzazione, per risolvere eventuali difetti persistenti e altre forme di problematiche tecniche (*technical debt)*. Tuttavia, la best practice è che nessuna funzionalità risulti completata fino a quando non sia stata integrata nel sistema e testata [Goucher09]. Un'altra buona pratica è quella di indirizzare i difetti rimanenti dalla precedente iterazione all'inizio della successiva iterazione, come parte del backlog per tale iterazione (chiamata anche "fix bug first", fissare prima i difetti). Tuttavia, alcuni lamentano che questa pratica porta ad una situazione in cui il lavoro totale che deve essere svolto nell'iterazione risulta non conosciuto e sarà più difficile stimare quando le funzionalità rimanenti potranno essere completate. Alla fine della sequenza di iterazioni, esiste una serie di attività di rilascio per ottenere il software pronto per il rilascio, anche se in alcuni casi il rilascio avviene al termine di ogni iterazione.

Quando viene usato il testing basato sul rischio come una delle strategie di test, un'analisi dei rischi di alto livello viene eseguita durante la pianificazione della release, e i tester spesso guidano tale analisi. Comunque, i rischi di qualità associati a ogni iterazione vengono individuati e valutati nella pianificazione dell'iterazione. Questa analisi del rischio può influenzare la sequenza di sviluppo, la priorità e la profondità del testing delle funzionalità. Influenza anche la stima dell'effort del test richiesto per ogni funzionalità (si veda Par.3.2).

In alcune pratiche Agile (ad es. Extreme Programming), viene utilizzato il pairing. Il pairing prevede che due tester lavorino insieme per testare una funzionalità. Il pairing può anche prevedere che un tester collabori con uno sviluppatore per implementare e testare una funzionalità. Il pairing può essere difficile quando il team di test è distribuito, ma i processi e gli strumenti possono aiutare a consentire il pairing distribuito. Per ulteriori informazioni sul lavoro distribuito, si veda [ISTQB_ALTM_SYL] (Par.2.8).

I tester possono anche essere dei coach per il testing e la qualità all'interno del team, condividendo la conoscenza sul testing e supportando le attività di quality assurance all'interno del team. Questo promuove un senso di "collective ownership" della qualità del prodotto.

Il test automation a tutti i livelli di test viene eseguito in molti team Agile e questo può significare che i tester impiegano del tempo per la creazione, esecuzione, monitoraggio e manutenzione dei test automatizzati e dei relativi risultati. A causa del utilizzo massiccio del test automation, una percentuale maggiore del testing manuale tende a essere eseguita sui progetti Agile utilizzando tecniche basate sull'esperienza e basate sui difetti, come attacchi software, testing esplorativo e error guessing (si veda [ISTQB_ALTA_SYL] Par.3.3 e 3.4 e [ISTQB_FL_SYL] Par.4.5). Mentre gli sviluppatori si focalizzeranno sulla creazione di unit test, i tester dovrebbero focalizzarsi sulla creazione dei test automatizzati di integrazione, di sistema e di integrazione di sistemi. Questo porta a una tendenza nei team Agile di favorire tester con un forte background tecnico e di test automation.

Un principio Agile fondamentale è che il cambiamento può verificarsi nel corso del progetto. Quindi, nei progetti Agile è preferibile una documentazione di prodotto leggera (leightweight). Modifiche alle funzionalità esistenti hanno implicazioni sul testing, in particolare implicazioni sul regression testing.

L'uso del test automation è un modo per gestire l'effort del test associato alle modifiche. Tuttavia, è importante che la frequenza di modifiche non superi la capacità del team di progetto di affrontare i rischi associati a tali modifiche.

2.1.2 Prodotti di Lavoro del Progetto

I prodotti di lavoro del progetto di immediato interesse per i tester Agile ricadono tipicamente in 3 categorie:

1. Prodotti di lavoro business-oriented, che descrivono cosa è richiesto (ad es. specifiche dei requisiti) e come usarlo (ad es. documentazione utente).
2. Prodotti di lavoro dello sviluppo, che descrivono come il sistema viene implementato (ad es. diagrammi entità–relazioni del database), cosa effettivamente implementa il sistema (ad es. il codice) o cosa valuta singoli pezzi di codice (ad es. unit test automatizzati).
3. Prodotti di lavoro del test, che descrivono come il sistema viene testato (ad es: strategia di test e test plan), cosa effettivamente testa il sistema (ad es. test manuali e automatizzati) o cosa descrive i risultati del test (ad es. dashboard di test come riportato nel par.2.2.1).

In un tipico progetto Agile, è pratica comune evitare di produrre grandi quantità di documentazione. Invece, il maggior focus è sull'avere un software funzionante, insieme a test automatizzati che dimostrino la conformità ai requisiti. Questo incoraggiamento a ridurre la documentazione si applica solo alla documentazione che non fornisce valore per il cliente. In un progetto Agile di successo, viene trovato un equilibrio tra l'aumento dell'efficienza, ottenuto riducendo la documentazione, e la necessità di fornire una documentazione sufficiente per supportare le attività di business, test, sviluppo e manutenzione. Il team deve prendere una decisione durante la pianificazione della release su quali prodotti di lavoro sono richiesti e quale livello di documentazione dei prodotti di lavoro è necessario.

Tipici prodotti di lavoro business-oriented nei progetti Agile includono le user story e i criteri di accettazione. Le user story sono il formato Agile delle specifiche dei requisiti e devono spiegare come il sistema si dovrebbe comportare rispetto ad una singola caratteristica o funzionalità consistente. Una user story dovrebbe definire una funzionalità sufficientemente piccola per essere completata in una singola iterazione. Insiemi più grandi di funzionalità correlate o un insieme di sotto-funzionalità che realizzano una singola funzionalità complessa, possono essere riferite come "epic". Le *epic* possono includere user story per differenti team di sviluppo. Ad esempio, una user story può descrivere quello che è richiesto a livello delle API (middleware) mentre un'altra user story descrive quello che è necessario a livello di User Interface (UI) (applicativo). Questi insiemi possono essere sviluppati durante una serie di sprint. Ogni *epic* e le sue user story dovrebbero essere associate a criteri di accettazione.

Tipici prodotti di lavoro dello sviluppo nei progetti Agile includono il codice. Gli sviluppatori Agile spesso creano anche unit test automatizzati. Questi test potrebbero essere creati dopo lo sviluppo del codice. In alcuni casi, tuttavia, gli sviluppatori creano i test in modo incrementale, prima di aver scritto ogni porzione di codice, per fornire un modo per verificare se funziona come previsto, una volta che la porzione di codice è stata scritta,. Questo approccio è indicato come test first development o test-driven development, ma in realtà i test hanno più una forma di specifiche di progettazione di basso livello eseguibili piuttosto che di test [Beck02].

Tipici prodotti di lavoro del test nei progetti Agile includono i test automatizzati, documenti quali i test plan, cataloghi dei rischi di qualità, test manuali, defect report e test log. I documenti vengono creati nel modo più leggero (lightweight) possibile, il che spesso è anche vero per questi documenti nei cicli di vita tradizionali. I tester produrranno anche metriche dei test dai defect report e dai test log, ponendo ancora enfasi su un approccio leggero.

In alcune implementazioni Agile, specialmente in progetti e prodotti soggetti a particolari normative, safety critical, distribuiti o altamente complessi, è richiesta un'ulteriore formalizzazione di questi prodotti di lavoro. Ad esempio, alcuni team trasformano le user story e i criteri di accettazione in specifiche dei requisiti più formali. Report di tracciabilità verticali e orizzontali possono essere preparati per soddisfare gli auditor, le normative e altri requisiti.

2.1.3 Livelli di Test

I livelli di test sono attività di test che sono logicamente correlate, spesso dalla maturità e dalla completezza dell'elemento sotto test.

Nei modelli del ciclo di vita sequenziali, i livelli di test sono spesso definiti in modo tale che i criteri di uscita di un livello sono parte dei criteri di ingresso per il livello successivo. In alcuni modelli iterativi, questa regola non si applica. I livelli di test si sovrappongono. Le specifiche dei requisiti, le specifiche di progettazione e le attività di sviluppo possono sovrapporsi ai livelli di test.

In alcuni cicli di vita Agile, la sovrapposizione si verifica perché le modifiche ai requisiti, alla progettazione e al codice possono accadere in qualsiasi momento in una iterazione. Mentre Scrum, in teoria, non consente modifiche alle user story dopo la pianificazione dell'iterazione, in pratica, a volte, tali modifiche si verificano. Durante un'iterazione, ogni user story progredirà normalmente in modo sequenziale attraverso le seguenti attività di test:

- Unit testing, tipicamente svolto dallo sviluppatore.
- Testing di accettazione della funzionalità, che talvolta è suddiviso in due attività:
  - Il testing di verifica della funzionalità riguarda il testing dei criteri di accettazione della user story, è spesso automatizzato e può essere eseguito dagli sviluppatori o dai tester.
  - Il testing di validazione della funzionalità è di solito manuale e può coinvolgere sviluppatori, tester e stakeholder di business nel lavorare e collaborare insieme per determinare se la funzionalità è pronta per l'utilizzo, per aumentare la visibilità dei progressi svolti e per ricevere un reale riscontro dagli stakeholder di business.

Inoltre, esiste spesso un processo parallelo di regression testing che avviene durante l'iterazione. Questo richiede la ri-esecuzione degli unit test automatizzati e dei test di verifica delle funzionalità dell'iterazione attuale e delle iterazioni precedenti, di solito attraverso un framework di Continuous Integration.

In alcuni progetti Agile, esiste un livello di test di sistema, che inizia una volta che la prima user story è pronta per tale testing. Questo può comportare l'esecuzione di test funzionali, di test non-funzionali per le prestazioni, l'affidabilità, l'usabilità e altri tipi di test rilevanti.

I team Agile possono utilizzare diversi tipi di testing di accettazione (usando il termine spiegato in [ISTQB_FL_SYL]). Possono essere eseguiti Alpha test interni e Beta test esterni al termine di ogni iterazione, al completamento di tutte le iterazioni o di una serie di iterazioni. Possono anche essere eseguiti User Acceptance Test, Operational Acceptance Test, test di accettazione contrattuale o normativo al termine di ogni iterazione, al completamento di tutte le iterazioni o di una serie di iterazioni .

2.1.4 Test Management e Configuration Management

I progetti Agile spesso richiedono un pesante utilizzo di strumenti automatizzati per sviluppare, testare e gestire lo sviluppo software. Gli sviluppatori utilizzano strumenti di analisi statica, strumenti di unit test e strumenti di copertura del codice. Gli sviluppatori rilasciano continuamente il codice e gli unit test in un Configuration Management System (attraverso il "check-in"), utilizzando build automatizzate e framework di test. Questi framework consentono il Continuous Integration di nuovo software nel sistema, l'analisi statica e gli unit test, eseguendoli ripetutamente man mano che viene eseguito il check in nel Configuration Management System di nuovo software [Kubaczkowski].

Questi test automatizzati possono includere anche test funzionali a livello di integrazione e di sistema. Questi test funzionali automatizzati possono essere creati utilizzando test harness funzionali, strumenti commerciali, oppure strumenti open-source per i test funzionali dell'interfaccia utente. Possono essere integrati con i test automatizzati eseguiti come parte del framework di Continuous Integration. In alcuni casi, a causa della loro durata, i test funzionali sono separati dagli unit test ed eseguiti meno

frequentemente. Ad esempio, gli unit test possono essere eseguiti ogni volta viene fatto il check-in di nuovo software, mentre i test funzionali più lunghi vengono eseguiti solo ogni pochi giorni.

Uno degli obiettivi dei test automatizzati è confermare che la build è funzionante e installabile. Se un test automatizzato fallisce, il team dovrebbe correggere il difetto in tempo per il successivo check-in del codice. Questo richiede un investimento per test report real-time che forniscano una buona visibilità dei risultati dei test. Questo approccio consente di ridurre i cicli costosi e inefficienti di "build-install-fail-rebuild-reinstall" che possono verificarsi in molti progetti tradizionali, poiché le modifiche che fanno fallire la build o causano errori nell'installazione software vengono rilevati rapidamente.

Gli strumenti per il test automation e l'automazione delle build aiutano a gestire il rischio di regressione associato ai frequenti cambiamenti che si verificano spesso nei progetti Agile. Tuttavia, l'eccessiva affidamento solo sugli unit test automatizzati per gestire questi rischi può essere un problema, poiché gli unit test spesso hanno un'efficacia limitata nel rilevamento dei difetti [Jones11]. Sono anche necessari test automatizzati a livello di integrazione e di sistema.

2.1.5 Scelte Organizzative per il Testing Indipendente

Come riportato in [ISTQB_FL_SYL], i tester indipendenti sono spesso più efficaci nella rilevazione dei difetti. In alcuni team Agile, gli sviluppatori creano molti dei test sottoforma di test automatizzati. Uno o più tester possono far parte del team, eseguendo diverse attività di test. Tuttavia, data la posizione di questi tester all'interno del team, esiste un rischio di perdita di indipendenza e di valutazione non obiettiva.

Altri team Agile mantengono team di test separati e completamente indipendenti, e assegnano tester a richiesta durante i giorni finali di ogni sprint. Questo può preservare l'indipendenza, e questi tester possono fornire una valutazione obiettiva e imparziale del software. Tuttavia, le pressioni sui tempi, la perdita di comprensione delle nuove funzionalità del prodotto e problemi di relazione con gli stakeholder di business e con gli sviluppatori portano spesso a problemi con questo approccio.

Una terza opzione è quella di avere un team di test separato e indipendente dove i tester sono assegnati al team Agile su una base a lungo termine dall'inizio del progetto, permettendo loro di mantenere la loro indipendenza pur acquisendo una buona conoscenza del prodotto e relazioni forti con gli altri membri del team. In aggiunta, il team di test indipendente può avere tester specializzati al di fuori del team Agile per lavorare su attività di lungo termine e/o indipendenti dall'iterazione, come lo sviluppo di strumenti di test automatizzati, l'esecuzione di test non-funzionali, la creazione e il supporto di ambienti e dati di test, e l'esecuzione di livelli di test che potrebbero non adattarsi bene all'interno di uno sprint (ad es. testing di integrazione di sistema).

2.2 Stato del Testing nei Progetti Agile

Una modifica viene gestita rapidamente nei progetti Agile. Una modifica implica che lo stato dei test, l'avanzamento dei test e la qualità del prodotto evolvono costantemente, e i tester devono trovare il modo per trasmettere tali informazioni al team, in modo da poter prendere le decisioni necessarie per rispettare i tempi previsti per il completamento di ogni iterazione con successo. Inoltre, la modifica può influenzare le funzionalità esistenti delle iterazioni precedenti. Pertanto, i test manuali e automatizzati devono essere aggiornati per affrontare in modo efficace il rischio di regressione.

2.2.1 Comunicare lo Stato del Testing, gli Avanzamenti e la Qualità del Prodotto

I team Agile progrediscono producendo software funzionante alla fine di ogni iterazione. Per determinare quando il team avrà un software funzionante, è necessario monitorare l'avanzamento di tutti i work item nell'iterazione e nella release. I tester nei team Agile utilizzano vari metodi per registrare l'avanzamento e lo stato dei test, inclusi i risultati del test automation, l'avanzamento dei task di test e delle user story, sulla task board Agile e sui grafici di burndown, che mostrano i progressi del team. Questi possono

quindi essere comunicati al resto del team utilizzando supporti quali dashboard wiki, e-mail secondo un formato di dashboard, o verbalmente durante gli stand-up meeting. I team Agile possono utilizzare strumenti che generano automaticamente gli status report in base ai risultati dei test e allo stato di avanzamento dell'attività, che a loro volta aggiorneranno le dashboard wiki o le e-mail. Questo metodo di comunicazione raccoglie anche metriche sul processo di test, che possono essere utilizzate nelle attività di miglioramento. Comunicare lo status del test in modo automatizzato aumenta anche il tempo a disposizione per i tester per concentrarsi sulla progettazione e l'esecuzione di più test case.

I team possono utilizzare i grafici burndown per monitorare i progressi su tutta la release e all'interno di ogni iterazione. Un grafico burndown [Crispin08] rappresenta la quantità di lavoro rimanente rispetto al tempo previsto per la release o l'iterazione

Per fornire una rappresentazione visiva istantanea e dettagliata dello stato attuale del whole-team, compreso lo stato del test, i team possono utilizzare le task board Agile. Le story card, i task di sviluppo, i task di test e gli altri task creati durante la pianificazione dell'iterazione (si veda par.1.2.5) vengono rappresentati sulla task board, spesso utilizzando carte di colore differente per identificare il tipo di task. Durante l'iterazione, l'avanzamento viene gestito attraverso lo spostamento di questi task tra le colonne della task board, che possono essere: *to do (da eseguire)*, *work in progress* (*in corso)*, (*verify*) (*in verifica)*, *done* (*eseguito*). I team Agile, per gestire e manutenere le story card e le task board, possono utilizzare strumenti che automatizzano dashboard e aggiornamenti di stato.

I task di test sulla task board riguardano i criteri di accettazione definiti per le user story. Quando gli script di test automation, i test manuali e i test esplorativi per un task di test raggiungono un cambiamento di stato, il task si sposta nella colonna *done* della task board. Il whole-team esamina lo stato della task board regolarmente, spesso durante gli stand-up meeting, per assicurarsi che i task si muovano sulla task board a una velocità accettabile. Se alcuni task (compresi i task di test) non si muovono o si muovono troppo lentamente, il team esegue la review e indirizza eventuali problemi che possono essere bloccanti per l'avanzamento di questi task.

Gli stand-up meeting quotidiani coinvolgono tutti i membri del team Agile, inclusi i tester. Durante il meeting, comunicano il loro stato attuale. L'agenda per ogni membro è [Agile Alliance Guide]:

- Cosa hai completato dall'ultimo meeting?
- Cosa hai pianificato di completare entro il prossimo meeting?
- Quali impedimenti stai incontrando?

Qualsiasi problema che possa bloccare l'avanzamento dei test viene comunicato durante il stand-up meeting, in modo che il whole-team sia consapevole dei problemi e di conseguenza li possa risolvere.

Per migliorare la qualità complessiva del prodotto, molti team Agile eseguono indagini (survey) sulla soddisfazione del cliente per ricevere feedback che il prodotto soddisfa le aspettative del cliente. I team possono utilizzare altre metriche similari a quelle ottenute nelle metodologie di sviluppo tradizionali, come il tasso dei test falliti/superati, i tassi di rilevamento dei difetti, i risultati dei test confermativi e dei regression test, la densità dei difetti, il numero di difetti rilevati e corretti, la copertura dei requisiti, la copertura dei rischi, la copertura del codice, e le modifiche al codice per migliorare la qualità del prodotto. Come in qualsiasi ciclo di vita, le metriche acquisite e comunicate devono essere rilevanti e aiutare il processo decisionale. Le metriche non devono essere utilizzate per premiare, punire o isolare un membro del team.

2.2.2 Gestire il Rischio di Regressioni Attraverso l'Evoluzione dei Test Case Manuali e Automatizzati

In un progetto Agile, il prodotto cresce man mano che ogni iterazione si completa. Pertanto, l'ambito del testing aumenta. Oltre a testare le modifiche del codice apportate nell'attuale iterazione, i tester devono anche verificare che nessuna regressione è stata introdotta sulle funzionalità che sono state sviluppate e testate nelle iterazioni precedenti. Il rischio di introdurre regressioni nello sviluppo Agile è elevato, a causa di modiche intensive del codice (linee di codice aggiunte, modificate o cancellate da una versione

all'altra). Dal momento che rispondere al cambiamento è un principio chiave Agile, le modifiche possono essere fatte anche su funzionalità precedentemente rilasciate per soddisfare esigenze di business. Per mantenere la velocità senza incorrere in una grande quantità di technical debt, è fondamentale che i team investano nel test automation a tutti i livelli di test il più presto possibile. É anche fondamentale che tutti gli asset del test, come i test automatizzati, i test case manuali, i dati di test e altri artefatti del test, siano mantenuti aggiornati ad ogni iterazione. E' altamente raccomandato che tutti gli asset del test siano mantenuti in uno strumento di Configuration Management per consentire il controllo delle versioni, per garantire facilità di accesso da parte di tutti i membri del team e per supportare le modifiche che devono essere eseguite a causa di modifiche della funzionalità, sempre conservando le informazioni storiche degli asset del test.

Poiché la ripetizione completa di tutti i test è possibile solo raramente, specialmente in progetti Agile con scadenze strette, i tester devono allocare tempo in ogni iterazione per eseguire la review dei test case manuali e automatzzati delle iterazioni precedenti (e della iterazione attuale), per selezionare i test case che possono essere candidati per una suite di regression test, e per eliminare i test case che non sono più rilevanti. I test scritti nelle prime iterazioni per verificare specifiche funzionalità possono avere poco valore nelle successive iterazioni, poiché modifiche alle funzionalità o nuove funzionalità possono modificare il comportamento di tali funzionalità.

Durante la review dei test case, i tester dovrebbero considerare l'adeguatezza per l'automazione. Il team deve automatizzare il maggior numero di test possibili dalle iterazioni precedenti e da quella attuale. Questo consente ai regression test automatizzati di ridurre il rischio di regressione con uno sforzo minore rispetto a quello che richiederebbe il regression testing manuale. Questo effort ridotto per i regression test consente ai tester di testare in modo più approfondito nuove caratteristiche e nuove funzionalità nell'iterazione attuale.

È fondamentale che i tester abbiano la capacità di identificare e aggiornare rapidamente i test case dalle iterazioni e/o release precedenti, che sono impattati dalle modifiche apportate nell'iterazione attuale. Durante la pianificazione della release dovrebbe essere definito come il team progetta, scrive e memorizza i test case. Devono essere adottate in anticipo buone pratiche per la progettazione e l'implementazione dei test, ed essere applicate in modo consistente. Tempi più brevi per il testing e modifiche costanti in ogni iterazione aumenteranno l'impatto di pratiche di progettazione e implementazione dei test scadenti.

L'uso del test automation, a tutti i livelli, consente ai team Agile di fornire un rapido feedback sulla qualità del prodotto. Test automatizzati ben scritti forniscono una documentazione dinamica delle funzionalità del sistema [Crispin08]. Attraverso la verifica dei test automatizzati e dei corrispondenti risultati dei test nel sistema di Configuration Management, allineato con il versioning delle build di prodotto, i team Agile possono eseguire la review della funzionalità testata e i risultati dei test per ogni build e in qualsiasi momento.

Gli unit test automatizzati vengono eseguiti prima che venga eseguito il check-in del codice sorgente nella mainline del Configuration Management System, per garantire che le modifiche al codice non portino a fallimenti nella build del software. Per ridurre i fallimenti della build, che possono rallentare il progresso del whole-team, non dovrebbe essere eseguito il check-in del codice a meno che tutti gli unit test automatizzati abbiano avuto esito positivo. I risultati degli unit test automatizzati forniscono un feedback immediato sul codice e sulla qualità della build, ma non sulla qualità del prodotto.

I test di accettazione automatizzati vengono eseguiti regolarmente come parte del Continuous Integration della build del sistema completo. Questi test vengono eseguiti su una build del sistema completo almeno quotidianamente, ma non sono in genere eseguiti ad ogni check-in del codice, poiché richiedono più tempo dell'esecuzione degli unit test automatizzati e potrebbero rallentare i check-in del codice. I risultati dei test di accettazione automatizzati forniscono un feedback sulla qualità del prodotto rispetto alla regressione dall'ultima build, ma non forniscono lo stato della qualità complessiva del prodotto.

I test automatizzati possono essere eseguiti continuamente sul sistema. Dovrebbe essere creato un sottoinsieme iniziale di test automatizzati per coprire le funzionalità del sistema e i punti di integrazione critici, subito dopo aver installato una nuova build nell'ambiente di test. Questi test sono comunemente

conosciuti come *build verification test*. I risultati dei build verification test forniranno un feedback immediato sul software dopo l'installazione, in modo che i team non perdano tempo a testare una build instabile.

I test automatizzati contenuti nell'insieme dei regression test sono in genere eseguiti come parte della build principale giornaliera nell'ambiente di Continuous Integration, e anche quando una nuova build viene installata nell'ambiente di test. Non appena un regression test automatizzato fallisce, il team si ferma e indaga le ragioni del fallimento del test. Il test può essere fallito a causa di modifiche funzionali previste nell'iterazione corrente, e in questo caso il test e/o la user story possono richiedere di essere aggiornati per riflettere i nuovi criteri di accettazione. In alternativa, il test può essere eliminato se un altro test è stato creato per coprire le modifiche. Tuttavia, se il test è fallito a causa di un difetto, è una buona pratica che il team corregga il difetto prima di progredire con nuove funzionalità.

Oltre al test automation, possono essere automatizzate le seguenti attività:

- Generazione dei dati di test
- Caricamento dei dati di test nei sistemi
- Installazione delle build negli ambienti di test
- Restore di una baseline dell'ambiente di test (ad es. il database o i file di dati di un sito web)
- Confronto dei dati di output

L'automazione di queste attività riduce il carico e consente al team di dedicare più tempo a sviluppare e testare nuove funzionalità.

2.3 Ruolo e Competenze di un Tester in un Team Agile

In un team Agile, i tester devono collaborare in modo stretto con tutti gli altri membri del team e con gli stakeholder di business. Questo ha una serie di implicazioni in termini di competenze che un tester deve avere e delle attività che svolge all'interno di un team Agile.

2.3.1 Competenze di un Tester Agile

I tester Agile dovrebbero avere tutte le competenze menzionate in [ISTQB_FL_SYL]. In aggiunta a queste competenze, un tester in un team Agile dovrebbe essere competente nel test automation, nel test-driven development, nell' acceptance test-driven development, nel testing white-box, black-box e basato sull'esperienza.

Poiché le metodologie Agile dipendono fortemente dalla collaborazione, dalla comunicazione e dall'interazione tra i membri del team e gli altri stakeholder al di fuori del team, i tester in un team Agile devono avere buone capacità relazionali. I tester in un team Agile dovrebbero:

- Essere positivi e orientati alla soluzione con i membri del team e gli stakeholder
- Mostrare un pensiero critico, orientato alla qualità e scettico verso il prodotto
- Acquisire attivamente informazioni dagli stakeholder (piuttosto che affidarsi completamente alle specifiche scritte)
- Valutare accuratamente e fornire report sui risultati e sull'avanzamento dei test, e sulla qualità del prodotto
- Lavorare in modo efficace con i rappresentanti del cliente e gli stakeholder per definire user story testabili, in particolare i criteri di accettazione
- Collaborare con il team, svolgere attività in coppia con i programmatori e gli altri membri del team

- Rispondere alle modifiche rapidamente, eventualmente modificando, aggiungendo o migliorando i test case
- Pianificare e organizzare il proprio lavoro.

La crescita continua delle competenze, inclusa la crescita delle competenze interpersonali, è essenziale per tutti i tester, inclusi quelli nei team Agile.

2.3.2 Il Ruolo di un Tester in un Team Agile

Il ruolo di un tester in un team Agile include attività che generano e forniscono feedback non solo sullo stato dei test, sull'avanzamento dei test e la qualità del prodotto, ma anche sulla qualità del processo. Oltre alle attività descritte in altri paragrafi di questo Syllabus, queste attività includono:

- Comprendere, implementare e aggiornare la strategia di test
- Misurare e fornire report sulla copertura del test rispetto a tutti i criteri di copertura applicabili
- Garantire l'utilizzo appropriato degli strumenti di test
- Configurare, utilizzare e gestire gli ambienti di test e i dati di test
- Segnalare i difetti e lavorare con il team per risolverli
- Essere coaching agli altri membri del team in aspetti rilevanti del test
- Garantire che appropriate attività di test siano schedulate durante la pianificazione della release e dell'iterazione
- Collaborare attivamente con gli sviluppatori e gli stakeholder di business per chiarire i requisiti, soprattutto in termini di testabilità, di consistenza e di completezza
- Partecipare in modo proattivo alle retrospettive del team, suggerendo e implementando miglioramenti

All'interno di un team Agile, ogni membro del team è responsabile della qualità del prodotto e gioca un ruolo nello svolgimento dei task relativi al test.

Le organizzazioni Agile possono incontrare alcuni rischi organizzativi relativi al test:

- I tester lavorano a così stretto contatto con gli sviluppatori da perdere il giusto mindset del tester
- I tester diventano tolleranti o silenti sulle pratiche inefficienti, inefficaci o di bassa qualità all'interno del team
- I tester non riescono tenere il passo con le modifiche richieste in iterazioni con vincoli di tempo stringenti.

Per mitigare questi rischi, le organizzazioni possono prendere in considerazione delle varianti per preservare l'indipendenza dei tester come discusso nel paragrafo 2.1.5.

3. Metodologie, Tecniche e Strumenti di Test Agile – 480 minuti

Termini:

Approccio di test, automazione dell'esecuzione dei test, criteri di accettazione, performance testing, regression testing, rischio di prodotto, rischio di qualità, stima del test, strategia di test, *Test Charter*, *Test-Driven Development*, *testing esplorativo*, *unit test framework*

Obiettivi di Apprendimento per Metodologie, Tecniche e Strumenti di Test Agile

3.1 Metodologie di Test Agile

FA-3.1.1 (K1) Richiamare i concetti Test-Driven Development, Acceptance Test-Driven Development e Behaviour-Driven development

FA-3.1.2 (K1) Richiamare i concetti della piramide di test

FA-3.1.3 (K2) Riassumere i quadranti del testing e le loro relazioni con i livelli di test e i tipi di test

FA-3.1.4 (K3) Per uno specifico progetto Agile, esercitarsi sul ruolo del tester in un team Scrum

3.2 Valutare i Rischi di Qualità e Stimare lo Sforzo del Testing

FA-3.2.1 (K3) Valutare i rischi di qualità all'interno di un progetto Agile

FA-3.2.2 (K3) Stimare lo sforzo del testing sulla base del contenuto dell'iterazione e dei rischi di qualità.

3.3 Tecniche nei Progetti Agile

FA-3.3.1 (K3) Interpretare le informazioni rilevanti per supportare le attività di test

FA-3.3.2 (K2) Spiegare ai clienti come definire criteri di accettazione testabili

FA-3.3.3 (K3) Data una user story, scrivere test case per l'Acceptance Test-Driven Development

FA-3.3.4 (K3) Scrivere test case, sia per il comportamento funzionale che non-funzionale, utilizzando le tecniche di test black-box a partire dalle user story.

FA-3.3.5 (K3) Eseguire il testing esplorativo per supportare il testing di un progetto Agile

3.4 Strumenti nei pProgetti Agile

FA-3.4.1 (K1) Rivalutare i vari strumenti disponibili per i tester in base alla loro finalità e alle attività nei progetti Agile

3.1 Metodologie di Test Agile

Esistono alcune pratiche di test che possono essere seguite in ogni progetto di sviluppo (Agile o non Agile) per produrre prodotti di qualità. Queste includono la scrittura anticipata dei test per descrivere un comportamento appropriato, il focalizzarsi sulla prevenzione, rilevazione e correzione anticipata dei difetti, e il garantire che i tipi giusti di test siano eseguiti al momento giusto e come parte del giusto livello

di test. I professionisti Agile mirano a introdurre queste pratiche quanto prima. I tester nei progetti Agile svolgono un ruolo chiave nel guidare l'uso di queste pratiche di test in tutto il ciclo di vita.

3.1.1 Test-Driven Development, Acceptance Test-Driven Development e Behaviour-Driven Development

Test-Driven Development, Acceptance Test-Driven Development e Behaviour-Driven Development sono tre tecniche complementari in uso tra i team Agile per eseguire il testing durante i diversi livelli di test. Ogni tecnica è un esempio di un principio fondamentale del test, il vantaggio del testing anticipato e delle attività di Quality Assurance (QA), perché i test vengono definiti prima che il codice sia scritto.

Test-Driven Development

Test-Driven Development è utilizzato per sviluppare codice guidato da test case automatizzati. Il processo per il Test-Driven Development comprende:

- Aggiungere un test che catturi l'idea del programmatore sul funzionamento desiderato di un piccolo pezzo di codice.
- Eseguire subito il test, che dovrebbe fallire pechè il codice non esiste ancora.
- Scrivere il codice ed eseguire il test, ripetendolo fino a quando il test non è stato superato.
- Eseguire il refactoring del codice dopo che il test è stato superato, rieseguire il test per assicurarsi che continui ad avere esito positivo con il codice modificato.
- Ripetere questo processo per il successivo piccolo pezzo di codice, eseguendo sia i test precedenti sia i test aggiunti.

I test scritti sono principalmente a livello di unit test e sono focalizzati sul codice, sebbene i test possano anche essere scritti a livello di integrazione o di sistema. Il Test-Driven Development ha raggiunto la sua popolarità attraverso Extreme Programming [Beck02], ma viene utilizzato anche in altre metodologie Agile e qualche volta in cicli di vita sequenziali. Aiuta gli sviluppatori a focalizzarsi sui risultati attesi chiaramente definiti. I test vengono automatizzati e vengono utilizzati nel Continuous Integration.

Acceptance Test-Driven Development

Acceptance Test-Driven Development definisce i criteri di accettazione e i test durante la creazione delle user story (si veda par.1.2.2). Acceptance Test-Driven Development è un approccio collaborativo che consente ad ogni stakeholder di comprendere come il componente software debba comportarsi, di cosa gli sviluppatori, i tester e i rappresentanti di business abbiano bisogno per garantire tale comportamento. Il processo di Acceptance Test-Driven Development è descritto nel par.3.3.2.

Acceptance Test-Driven Development crea test riutilizzabili per il regression testing. Strumenti specifici supportano la creazione e l'esecuzione di tali test, spesso nell'ambito del processo di Continuous Integration. Questi strumenti possono connettersi ai service layer e ai data layer dell'applicazione, permettendo di eseguire i test a livello di sistema o di accettazione. Acceptance Test-Driven Development consente una rapida risoluzione dei difetti e la validazione del comportamento delle funzionalità. Aiuta a determinare se i criteri di accettazione per la funzionalità sono soddisfatti.

Behaviour-Driven Development

Behaviour-Driven Development [Chelimsky10] permette allo sviluppatore di focalizzarsi sul testing del codice in base al comportamento atteso del software. Poiché i test sono basati sul comportamento esibito dal software, i test sono generalmente più facili da comprendere per gli altri membri del team e per gli altri stakeholder.

Possono essere usati specifici framework di Behaviour-Driven Development per definire i criteri di accettazione in base al formato *dato/quando/allora (given/when/then)*:

dato il contesto iniziale,

quando avviene un evento,

allora si assicura un risultato.

Da tali requisiti, il framework di Behaviour-Driven Development genera codice che può essere utilizzato dagli sviluppatori per creare i test case. Behaviour-Driven Development aiuta lo sviluppatore a collaborare con gli altri stakeholder, inclusi i tester, per definire unit test accurati focalizzati sulle esigenze di business.

3.1.2 La Piramide di Test

Un sistema software può essere testato a diversi livelli. Livelli di test tipici sono, dalla base della piramide verso l'alto, unit test, test di integrazione, di sistema e di accettazione (si veda [ISTQB_FL_SYL], Par.2.2). La piramide di test sottolinea che si avranno un gran numero di test ai livelli più bassi (base della piramide) e, come lo sviluppo si sposta verso i livelli superiori, il numero di test diminuisce (parte superiore della piramide). Di solito gli unit test e i test a livello di integrazione vengono automatizzati e vengono creati utilizzando strumenti basati su API. A livello di sistema e di accettazione, i test automatizzati vengono creati utilizzando strumenti basati sulla GUI. Il concetto di piramide di test si basa sul principio di QA e testing anticipato (cioè, eliminando i difetti il più presto possibile nel ciclo di vita).

3.1.3 Quadranti del Testing, Livelli di Test e Tipi di Test

I quadranti del testing, definiti da Brian Marick [Crispin08], allineano i livelli di test agli appropriati tipi di test nella metodologia Agile. Il modello dei quadranti del testing, e le sue varianti, aiuta a garantire che tutti i tipi di test e tutti i livelli di test importanti siano inclusi nel ciclo di vita dello sviluppo. Questo modello fornisce anche un modo per differenziare e descrivere i tipi di test per tutti gli stakeholder, tra cui sviluppatori, tester e rappresentanti di business.

Nei quadranti del testing, i test possono essere orientati al business (utente) o alla tecnologia (sviluppatore). Alcuni test supportano il lavoro svolto dal team Agile e verificano il comportamento del software. Altri test possono verificare il prodotto. I test possono essere completamente manuali, completamente automatizzati, una combinazione di manuali ed automatizzati, oppure manuali ma supportati da strumenti. I quattro quadranti sono i seguenti:

- Quadrante Q1: è a livello di unit test, orientato alla tecnologia e supporta gli sviluppatori. Questo quadrante contiene unit test. Questi test dovrebbero essere automatizzati e inclusi nel processo di Continuous Integration.
- Quadrante Q2: è a livello di test di sistema, orientato al business e verifica il comportamento del prodotto. Questo quadrante contiene test funzionali, esempi, test delle user story, prototipi di user experience e simulazioni. Questi test verificano i criteri di accettazione e possono essere manuali o automatizzati. Sono spesso creati durante lo sviluppo della user story e quindi migliorano la qualità delle user story. Sono utili per la creazione di suite di regression test automatizzate.
- Quadrante Q3: è il livello di test di sistema o user acceptance test, orientato al business e contiene i test che criticano il prodotto, utilizzando scenari e dati realistici. Questo quadrante contiene test esplorativi, scenari, flussi di processo, test di usabilità, user acceptance test, alpha test e beta test. Questi test sono spesso manuali e sono user-oriented.
- Quadrante Q4: è il livello di test di sistema o operational acceptance test, orientato alla tecnologia e contiene i test che criticano il prodotto. Questo quadrante contiene performance test, load test, stress test, test di scalabilità, di sicurezza, di manutenzione, di gestione della memoria, di compatibilità, di interoperabilità, di migrazione dei dati, di infrastruttura e di recovery. Questi test sono spesso automatizzati.

Durante ogni iterazione, possono essere richiesti i test di alcuni o di tutti i quadranti. I quadranti del testing si applicano al testing dinamico e al testing statico.

3.1.4 Il Ruolo di un Tester

In questo Syllabus, è stato fatto un riferimento generale ai metodi e alle tecniche Agile, e al ruolo di un tester all'interno dei diversi cicli di vita Agile. Questo paragrafo esamina specificamente il ruolo di un tester in un progetto che segue un ciclo di vita Scrum [Aalst13].

Lavoro di Squadra (Teamwork)

Il lavoro di squadra è un principio fondamentale nello sviluppo Agile. Agile enfatizza l'approccio whole-team che consiste di sviluppatori, tester e rappresentanti di business che lavorano insieme. Di seguito sono riportate le migliori best practice organizzative e comportamentali nei team Scrum:

- Cross-funzionale: Ogni membro del team porta un insieme differente di competenze. Il team lavora insieme sulla strategia di test, sulla pianificazione dei test, sulla specifica dei test e sull'esecuzione dei test, sulla valutazione dei test, e sulla reportistica dei risultati dei test.
- Auto-organizzato: Il team può essere composto solo di sviluppatori, ma, come descritto nel paragrafo 2.1.5, l'ideale sarebbe che fossero presenti uno o più tester.
- Colocato: I tester lavorano fisicamente insieme agli sviluppatori e al Product Owner.
- Collaborativo: I tester collaborano con gli altri membri del team, con gli altri team, gli stakeholder, il Product Owner e lo Scrum Master
- Responsabile (Empowered): Le decisioni tecniche sulla progettazione e il testing vengono prese dal team completo (sviluppatori, tester e Scrum Master), in collaborazione con il Product Owner e gli altri team se necessario.
- Impegnato (Committed): Il tester ha l'impegno di mettere in discussione e valutare il comportamento e le caratteristiche del prodotto rispetto alle aspettative e alle esigenze dei clienti e degli utenti.
- Trasparente: L'avanzamento dello sviluppo e del testing è visibile sulla task board Agile (si veda par.2.2.1).
- Credibile: Il tester deve garantire la credibilità della strategia per il testing, la sua implementazione ed esecuzione, altrimenti gli stakeholder non si fideranno dei risultati dei test. Questo è spesso eseguito fornendo informazioni agli stakeholder sul processo di test.
- Aperto ai feedback: Il feedback è un aspetto importante per avere successo in ogni progetto, in particolare nei progetti Agile. Le retrospettive consentono ai team di imparare dai successi e dai fallimenti.
- Resiliente: Il testing deve essere in grado di rispondere al cambiamento, come tutte le altre attività nei progetti Agile.

Queste best practice massimizzano la probabilità di un testing efficace nei progetti Scrum.

Sprint Zero

Lo Sprint zero è la prima iterazione del progetto in cui hanno luogo molte attività di preparazione (si veda par.1.2.5). Durante questa iterazione il tester collabora con il team sulle seguenti attività:

- Identificare l'ambito del progetto (cioè il Product Backlog).
- Creare un'architettura iniziale del sistema e dei prototipi di alto livello
- Pianificare, acquisire e installare gli strumenti necessari (ad es. per test management, defect management, test automation e Continuous Integration)
- Creare una strategia di test iniziale per tutti i livelli di test, indirizzando (tra gli altri aspetti) l'ambito del test, i rischi tecnici, i tipi di test (si veda par.3.1.3) e gli obiettivi di copertura
- Eseguire una analisi iniziale dei rischi di qualità (si veda par.3.2.1)

- Definire le metriche di test per misurare il processo di test, l'avanzamento del testing nel progetto e la qualità del prodotto
- Specificare la Definition of Done
- Creare la task board (si veda par.2.2.1)
- Definire quando continuare o fermare il testing prima di rilasciare il sistema al cliente.

Lo sprint zero indica la direzione per gli obiettivi che il testing deve raggiungere e come raggiungerlo durante gli sprint

Integrazione

Nei progetti Agile, l'obiettivo è quello di rilasciare valore al cliente su base continuativa (preferibilmente in ogni sprint). A tal fine, la strategia di integrazione dovrebbe considerare sia la progettazione che il testing. Per attivare una strategia di test continuo per le funzionalità e caratteristiche da consegnare, è importante identificare tutte le dipendenze tra funzioni e caratteristiche sottostanti.

Pianificazione dei Test

Poiché il testing è completamente integrato nel team Agile, la pianificazione dei test dovrebbe iniziare durante la sessione di pianificazione della release ed essere aggiornata durante ogni sprint. La pianificazione dei test per la release e per ogni sprint dovrebbe indirizzare i problemi discussi nel paragrafo 1.2.5.

La pianificazione dello sprint si traduce in una serie di task da inserire nella task board, dove ogni task dovrebbe avere una lunghezza di uno o due giorni di lavoro. Inoltre, eventuali problematiche del testing dovrebbero essere tracciate per mantenere un flusso costante del testing.

Pratiche di Test Agile

Molte pratiche possono essere utili per i tester di un team Scrum, alcune delle quali comprendono:

- Lavoro in coppia (pairing): Due membri del team (ad es. un tester e uno sviluppatore, due tester, un tester e un product owner) si siedono vicini davanti ad una workstation per eseguire un testing o altri task dello sprint.
- Progettazione dei test incrementale: I test case e i Test Charter sono creati gradualmente dalle user story e da altre basi di test, a partire da test semplici e spostandosi verso test più complessi.
- Mappe mentali: Le mappe mentali sono uno strumento utile per il testing [Crispin08]. Ad esempio, i tester possono usare le mappe mentali per identificare quali sessioni di test eseguire, per visualizzare le strategie di test e per descrivere i dati di test.

Queste pratiche sono in aggiunta alle altre pratiche discusse in questo Syllabus e nel capitolo 4 di [ISTQB_FL_SYL].

3.2 Valutare i Rischi di Qualità e Stimare l'Effort del Test

Un tipico obiettivo del test in tutti i progetti, Agile o tradizionali, è quello di ridurre il rischio di problemi di qualità del prodotto ad un livello accettabile prima del rilascio. I tester nei progetti Agile possono utilizzare gli stessi tipi di tecniche utilizzate nei progetti tradizionali per individuare i rischi di qualità (o rischi di prodotto), valutare il livello di rischio associato, stimare l'effort richiesto per ridurre sufficientemente questi rischi, e quindi mitigare questi rischi attraverso la progettazione, l'implementazione e l'esecuzione dei test. Tuttavia, le iterazioni brevi e la frequenza delle modifiche nei progetti Agile richiedono alcuni adattamenti di queste tecniche.

3.2.1 Valutare i Rischi di Qualità nei Progetti Agile

Una delle tante sfide nel testing è la selezione, allocazione e prioritizzazione appropriata delle condizioni di test. Questo include la valutazione della quantità appropriata di effort da allocare per coprire ogni condizione con i test, e mettere in sequenza i test risultanti in modo da ottimizzare l'efficacia e l'efficienza delle attività di test da eseguire. Le strategie di identificazione, analisi e mitigazione del rischio possono essere usate dai tester nei team Agile per aiutare a determinare un numero accettabile di test case da eseguire, anche se molti vincoli e variabili che interagiscono tra loro possono richiedere dei compromessi.

Il rischio è la possibilità che accada un risultato o un evento negativo o non desiderato. Il livello di rischio è ottenuto valutando la probabilità di accadimento del rischio ed il suo impatto. Quando l'effetto primario del potenziale problema è la qualità del prodotto, i problemi potenziali sono indicati come rischi di qualità o rischi di prodotto. Quando l'effetto primario del potenziale problema è il successo del progetto, i potenziali problemi sono indicati come rischi di progetto o rischi di pianificazione [Black07] [vanVeenendaal12].

Nei progetti Agile, l'analisi dei rischi di qualità si svolge in due occasioni:

- Pianificazione della release: i rappresentanti di business che conoscono le funzionalità della release forniscono una overview di alto livello dei rischi, e il team completo, inclusi i tester, può aiutare nella identificazione e nella valutazione dei rischi.
- Pianificazione dell'iterazione: il team completo identifica e valuta i rischi di qualità.

Esempi di rischi di qualità per un sistema includono:

- Calcoli errati nei report (rischio funzionale relativo all'accuratezza).
- Risposta lenta agli input dell'utente (rischio non-funzionale relativo all'efficienza e ai tempi di risposta).
- Difficoltà di comprensione delle schermate e dei campi dati (rischio non-funzionale relativo all'usabilità e alla comprensibilità).

Come menzionato in precedenza, un'iterazione inizia con la pianificazione dell'iterazione, che culmina in task stimate sulla task board. Questi task possono essere parzialmente prioritizzate in base al livello di rischio di qualità ad esse associato. Task associati ai rischi più elevati dovrebbero iniziare prima e richiedere più effort di test. Task associati ai rischi più bassi dovrebbero iniziare più tardi e richiedere meno effort di test.

Un esempio di come il processo di analisi dei rischi di qualità in un progetto Agile possa essere eseguito durante la pianificazione dell'iterazione è descritto nei seguenti passi:

1. Riunire i membri del team Agile, inclusi i tester.
2. Elencare tutti gli elementi del backlog per l'iterazione attuale (ad es. sulla task board).
3. Identificare i rischi di qualità associati ad ogni elemento, considerando tutte le caratteristiche di qualità rilevanti.
4. Valutare ogni rischio identificato, che include due attività: categorizzare il rischio e determinare il livello di rischio in base all'impatto e alla probabilità dei difetti.
5. Determinare l'estensione del testing proporzionale al livello di rischio.
6. Selezionare appropriate tecniche di test per mitigare ogni rischio, in base al rischio, al livello di rischio e alle caratteristiche di qualità rilevanti.

Il tester poi progetta, implementa ed esegue i test per mitigare i rischi. Questo include la totalità delle funzionalità, dei comportamenti, delle caratteristiche di qualità e degli attributi che influenzano la soddisfazione del cliente, dell'utente e degli stakeholder.

Nel corso del progetto, il team dovrebbe rimanere a conoscenza di informazioni aggiuntive che possano modificare l'insieme dei rischi e/o il livello di rischio associato ai rischi di qualità conosciuti. Dovrebbero essere eseguito un adeguamento periodico dell'analisi dei rischi di qualità, che si traduce in adeguamenti dei test. Gli adeguamenti includono l'identificazione di nuovi rischi, la rivalutazione del livello dei rischi esistenti e la valutazione dell'efficacia delle attività di mitigazione del rischio.

I rischi di qualità possono essere mitigati anche prima dell'inizio dell'esecuzione dei test. Ad esempio, se vengono rilevati problemi con le user story durante l'identificazione del rischio, il team di progetto può eseguire una review approfondita delle user story come strategia di mitigazione.

3.2.2 Stima del Testing Basata sui Contenuti e sul Rischio

Durante la pianificazione della release, il team Agile stima l'effort richiesto per completare la release. La stima riguarda anche l'effort per il testing. Una tecnica di stima comune utilizzata nei progetti Agile è il planning poker, una tecnica basata sul consenso. Il Product Owner o il cliente legge una user story a chi deve effettuare la stima. Ogni stimatore ha un mazzo di carte con valori similari alla sequenza di Fibonacci (cioè, 0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...) o qualsiasi altra progressione di scelta (ad es. le misure di una camicia da extra-small a extra-extra-large). I valori rappresentano il numero di story point, di giorni di effort o altre unità in cui il team effettua la stima. La sequenza di Fibonacci è raccomandata perché i numeri nella sequenza riflettono l'incertezza che cresce proporzionalmente con la dimensione della user story. Una stima alta di solito significa che la user story non è ben compresa o dovrebbe essere suddivisa in user story più piccole.

Gli stimatori discutono la funzionalità e pongono domande al Product Owner, se necessario. Aspetti come l'effort dello sviluppo e del testing, la complessità della user story e l'ambito del testing giocano un ruolo nella stima. Pertanto, è opportuno includere il livello di rischio di un elemento del backlog, oltre alla priorità specificata dal Product Owner, prima di iniziare la sessione di planning poker. Quando la funzione è stata discussa in modo completo, ogni stimatore seleziona, senza mostrarla agli altri, una carta per indicare la sua stima. Tutte le carte vengono poi scoperte nello stesso momento. Se tutti gli stimatori hanno selezionato lo stesso valore, questo diventa la stima. In caso contrario, gli stimatori discutono le differenze nelle stime, dopo di che il giro di poker viene ripetuto fino a quando viene raggiunto un accordo, sia per consenso o per l'applicazione di regole (ad es., uso della mediana o del punteggio più alto) per limitare il numero di giri di poker. Queste discussioni assicurano una stima affidabile dell'effort necessario per completare gli elementi del Product Backlog richiesti dal Product Owner e contribuiscono a migliorare la conoscenza collettiva di ciò che deve essere fatto [Cohn04].

3.3 Tecniche nei Progetti Agile

Molte delle tecniche di test e dei livelli di test che si applicano ai progetti tradizionali possono essere applicate anche ai progetti Agile. Tuttavia, per i progetti Agile, dovrebbero essere fatte alcune considerazioni specifiche e delle variazioni alle tecniche di test, alla terminologia e alla documentazione.

3.3.1 Criteri di Accettazione, Copertura Adeguata e altre Informazioni sul Testing

I progetti Agile delineano all'inizio del progetto i requisiti iniziali come user story in un backlog prioritizzato. I requisiti iniziali sono brevi e di solito seguono un formato predefinito (si veda par.1.2.2). Sono anche importanti i requisiti non-funzionali, come usabilità e prestazioni, e possono essere specificati come singole user story oppure essere collegati ad altre user story funzionali. I requisiti non-funzionali possono seguire un formato predefinito o standard, come [ISO25000], oppure standard specifici di settore.

Le user story servono come importante base di test. Altre importanti basi per il test includono:

- Esperienze da progetti precedenti

- Funzioni, funzionalità e caratteristiche di qualità esistenti del sistema.
- Codice, architettura e progettazione
- Profili utente (contesto, configurazioni di sistema e comportamento dell'utente)
- Informazioni sui difetti dal progetto attuale o dai precedenti
- Categorizzazione dei difetti in una tassonomia dei difetti
- Standard applicabili (ad es. [DO-178B] per il software avionico)
- Rischi di qualità (si veda par.3.2.1).

Durante ogni iterazione, gli sviluppatori creano codice che implementa le funzioni e le funzionalità descritte nelle user story, con le caratteristiche di qualità rilevanti, e questo codice viene verificato e validato attraverso il testing di accettazione. Per essere testabili, i criteri di accettazione dovrebbero indirizzare i seguenti aspetti, dove applicabili [Wiegers13]:

- Comportamento funzionale: Il comportamento osservabile dall'esterno con azioni utente, come alcuni input che operano sotto specifiche configurazioni.
- Caratteristiche di qualità: Come il sistema esegue il comportamento specificato. Le caratteristiche possono anche essere indicate come attributi di qualità o requisiti non-funzionali. Caratteristiche di qualità tipiche sono le prestazioni, l'affidabilità, l'usabilità, ecc.
- Scenari (use case): Una sequenza di azioni tra un attore esterno (spesso un utente) e il sistema per raggiungere un obiettivo o completare un task di business.
- Regole di business: Attività che possono essere eseguite nel sistema solo sotto certe condizioni definite da vincoli e procedure esterne (ad es. le procedure usate dalle compagnie di assicurazione per gestire reclami).
- Interfacce esterne: Descrizioni delle connessioni tra il sistema da sviluppare e il mondo esterno. Le interfacce esterne possono essere suddivise in tipi differenti (interfacce utente, interfacce con altri sistemi, ecc.).
- Vincoli: Qualsiasi vincolo di progettazione o implementazione che restringeranno le opzioni per lo sviluppatore. I dispositivi con software embedded spesso devono rispettare vincoli fisici come le dimensioni, il peso e le connessioni di interfaccia.
- Definizione dei dati: Il cliente può descrivere il formato, il tipo di dati, i valori ammessi e i valori di default per un elemento di dato nella composizione di una struttura dati di business complessa (ad es., il codice di avviamento postale in un indirizzo).

In aggiunta alle user story e ai criteri di accettazione associati, per il tester sono rilevanti altre informazioni, che includono:

- Come il sistema si supponga funzioni e sia utilizzato.
- Le interfacce di sistema che possono essere utilizzate/accedute per testare il sistema.
- Se il supporto dello strumento attuale è sufficiente.
- Se il tester ha sufficiente conoscenze e competenze per effettuare i test necessari.

I tester spesso scoprono la necessità di ulteriori informazioni (ad es. la copertura del codice) durante le iterazioni e dovrebbero lavorare in modo collaborativo con il resto dei membri del team Agile per ottenere tali informazioni. Le informazioni rilevanti giocano un ruolo nel determinare se un particolare task può essere considerato DONE. Questo concetto di Definition of Done è critica nei progetti Agile e si applica in un numero di modi diversi come discusso nei seguenti sottoparagrafi.

Livello di Test

Ogni livello di test ha la propria Definition of Done. La seguente lista fornisce esempi che possono essere rilevanti per i differenti livelli di test:

- Unit testing
  - 100% di copertura delle decisioni dove possibile, con review accurate di ogni cammino non fattibile
  - Analisi statica eseguita su tutto il codice
  - Nessun difetto importante non risolto (classificato in base alla priorità e alla severità)
  - Nessun technical debt conosciuto e non accettabile rimanente nella progettazione e nel codice [Jones11]
  - Review eseguita su tutto il codice, gli unit test e i risultati degli unit test
  - Tutti gli unit test automatizzati
  - Caratteristiche importanti entro i limiti previsti (ad es. le prestazioni)
- Testing di integrazione
  - Tutti i requisiti funzionali testati, inclusi i test positivi e negativi, con il numero di test calcolato in base alle dimensioni, alla complessità e ai rischi.
  - Tutte le interfacce tra i componenti testate.
  - Tutti i rischi di qualità coperti in base all'estensione concordata del testing.
  - Nessun difetto importante non risolto (prioritizzato in base al rischio e all'importanza)
  - Tutti i difetti rilevati sono stati tracciati
  - Quando possibile, tutti i regression test automatizzati e memorizzati in un repository comune.
- Testing di sistema
  - Test end-to-end delle user story, delle funzionalità e delle funzioni.
  - Copertura di tutte le Personas dell'utente
  - Le più importanti caratteristiche di qualità del sistema coperte (ad es. prestazioni, robustezza, affidabilità).
  - Testing eseguito in un ambiente simile alla produzione, compreso tutto l'hardware e il software per tutte le configurazioni supportate, per quanto possibile.
  - Tutti i rischi di qualità coperti in base all'estensione concordata del testing.
  - Quando possibile, tutti i regression test automatizzati e memorizzati in un repository comune.
  - Tutti i difetti rilevati sono stati tracciati e possibilmente corretti.
  - Nessun difetto importante non risolto (prioritizzato in base al rischio e all'importanza)

User Story

La Definition of Done per le user story può essere determinata dai seguenti criteri:

- Le user story selezionate per l'iterazione sono complete, comprese dal team e hanno criteri di accettazione dettagliati e testabili.

- Tutti gli elementi della user story sono stati specificati, sottoposti a review, inclusi i test di accettazione della user story, e sono stati completati.
- I task necessari per implementare e testare le user story selezionate sono stati identificati e stimati dal team.

Funzionalità

La Definition of Done per le funzionalità, che possono estendersi a più user story o epic, può includere:

- Tutte le user story che la compongono, con i relativi criteri di accettazione, sono definite e approvate dal cliente.
- La progettazione è completa, con nessun technical debt.
- Il codice è completo, con nessun technical debt conosciuto o refactoring non completato.
- Sono stati eseguiti gli unit test e sono stati raggiunti i livelli di copertura definiti.
- I test di integrazione e di sistema per la funzionalità sono stati eseguiti in base ai criteri di copertura definiti.
- Nessun difetto importante deve essere ancora corretto.
- La documentazione della funzionalità è completa, e può includere release note, i manuali utente e le funzioni di help on line.

Iterazione

La Definition of Done per l'iterazione può includere:

- Tutte le funzionalità dell'iterazione sono pronte e testate singolarmente secondo i criteri a livello di funzionalità
- Ogni difetto non critico, che non può essere corretto all'interno dei vincoli dell'iterazione, è stato aggiunto al Product Backlog e prioritizzato.
- L'integrazione di tutte le funzionalità dell'iterazione è stata completata e testata.
- La documentazione è stata scritta, sottoposta a review e approvata.

A questo punto, il software è potenzialmente rilasciabile perché l'iterazione è stata completata con successo, ma non tutte le iterazioni comportano un rilascio.

Release

La Definition of Done per una release, che può estendersi a più iterazioni, può includere le seguenti aree:

- Copertura: Tutti gli elementi rilevanti della base di test per tutti i contenuti della release sono state coperti dal testing. L'adeguatezza della copertura è determinata da quello che è nuovo o modificato, dalla sua complessità e dalla dimensione, e dai rischi di failure associati.
- Qualità: L'intensità dei difetti (ad es., quanti difetti sono rilevati al giorno o per transazione), la densità dei difetti (ad es., numero di difetti rilevati rispetto al numero di user story, all'effort, e/o agli attributi di qualità), il numero stimato dei difetti rimanenti sono all'interno di limiti accettabili, le conseguenze di difetti irrisolti e rimanenti (ad es., severità e priorità) sono compresi ed accettabili, il livello residuo di rischio associato a ogni rischio di qualità identificato è stato compreso ed è accettabile.
- Tempi: Se la data di rilascio prevista è stata raggiunta, devono essere considerate le valutazioni di business associate al rilascio o al mancato rilascio del software.
- Costi: I costi stimati del ciclo di vita dovrebbero essere utilizzati per calcolare il ritorno di investimento del sistema rilasciato (cioè, i costi calcolati per lo sviluppo e la manutenzione

dovrebbero essere notevolmente inferiori rispetto alle vendite totali previste del prodotto). La parte principale del costo del ciclo di vita è spesso dovuto alla manutenzione dopo il rilascio del prodotto, a causa del numero di difetti rilasciati in produzione.

3.3.2 Applicare Acceptance Test-Driven Development

Acceptance Test-Driven Development è un approccio test-first. I test case vengono creati prima di implementare la user story. I test case vengono creati dal team Agile, che include lo sviluppatore, il tester, ed i rappresentanti di business [Adzic09] e possono essere manuali o automatizzati. Il primo passo è un workshop specifico in cui la user story viene analizzata, discussa e scritta dagli sviluppatori, dai tester e dai rappresentanti di business. Qualsiasi incompletezza, ambiguità o errore nella user story viene risolta durante questo processo.

Il passo successivo è quello di creare i test. Questo può essere fatto insieme dal team o singolarmente dal tester. In ogni caso, una persona indipendente come un rappresentante di business valida i test. I test sono esempi che descrivono le caratteristiche specifiche della user story. Questi esempi aiuteranno il team a implementare correttamente la user story. Poiché esempi e test sono gli stessi, questi termini sono spesso interscambiati. Il lavoro inizia con esempi di base e domande aperte.

Tipicamente, i primi test sono i test positivi che confermano il corretto comportamento senza eccezioni o condizioni di errore, compresa la sequenza delle attività eseguite se tutto funziona come previsto. Dopo aver eseguito i test dei cammini positivi, il team dovrebbe scrivere i test dei cammini negativi e coprire anche gli attributi non-funzionali (ad es. prestazioni, usabilità). I test sono espressi in modo che ogni stakeholder sia in grado di comprenderli, contengono frasi scritte nel linguaggio naturale con le necessarie precondizioni, se necessarie, gli input e i relativi output.

Gli esempi devono coprire tutte le caratteristiche della user story e non dovrebbero aggiungere nulla alla user story. Questo significa che non deve esistere un esempio che descriva un aspetto della user story non documentato nella user story stessa. Inoltre, non dovrebbero esistere due esempi che descrivano le stesse caratteristiche della user story.

3.3.3 Progettazione dei Test Black-Box Funzionali e Non-Funzionali

Nel testing Agile, molti test vengono creati dai tester in parallelo alle attività di codifica degli sviluppatori. Proprio come gli sviluppatori codificano in base alle user story e ai criteri di accettazione, anche i tester creano i test in base alle user story e ai loro criteri di accettazione. (Alcuni test, come i test esplorativi e alcuni altri test basati sull'esperienza, vengono creati più tardi, durante l'esecuzione del test, come spiegato nel par.3.3.4.) I tester possono applicare tecniche tradizionali di progettazione dei test black-box, come il partizionamento di equivalenza, l'analisi ai valori limite, il testing della tabella delle decisioni e il testing delle transizioni di stato. Ad esempio, analisi ai valori limite potrebbe essere utilizzata per selezionare i valori di test quando un cliente è limitato nel numero di elementi che può selezionare per l'acquisto.

In molte situazioni, i requisiti non-funzionali possono essere documentati come user story. Le tecniche di progettazione dei test black-box (come l'analisi ai valori limite) possono anche essere usate per creare test per le caratteristiche di qualità non-funzionali. La user story potrebbe contenere requisiti di prestazione o di affidabilità. Ad esempio, una data esecuzione non può superare un limite di tempo, o un dato numero di operazioni può fallire entro uno specifico numero di volte.

Per altre informazioni sull'uso delle tecniche progettazione dei test black-bix, si veda [ISTQB_ALTA_SYL].

3.3.4 Testing Esplorativo e Testing Agile

Il testing esplorativo è importante nei progetti Agile a causa del tempo limitato a disposizione per l'analisi dei test e i dettagli limitati delle user story. Per ottenere i migliori risultati, il testing esplorativo dovrebbe essere combinato con altre tecniche basate sull'esperienza come parte di una strategia di test reattiva, miscelate con altre strategie di test, come il testing analitico basato sul rischio, il testing analitico basato

sui requisiti, il testing model-based e il testing avverso alla regressione. Le strategie di test e l'uso di strategie di test miste sono discussi in [ISTQB_FL_SYL].

Nel testing esplorativo, la progettazione e l'esecuzione dei test avvengono allo stesso tempo, guidati da un Test Charter preparato. Un Test Charter fornisce le condizioni di test da coprire durante una sessione di test time-boxed. Durante il testing esplorativo, i risultati dei test più recenti guidano il test successivo. La stesse tecniche white-box e black-box possono essere utilizzate per progettare i test, come quando si esegue il testing pre-progettato.

Un Test Charter può includere le seguenti informazioni:

- Attore: l'utente del sistema
- Scopo: il tema del Test Charter, incluso quale particolare obiettivo l'attore vuole ottenere, cioè le condizioni di test.
- Setup: cosa deve essere pronto per iniziare l'esecuzione dei test
- Priorità: importanza relativa del Test Charter, in base alla priorità associata alla user story o al livello di rischio.
- Riferimenti: specifiche (ad es. user story), rischi o altre fonti di informazione.
- Dati: quali dati sono necessari per eseguire il Test Charter
- Attività: una lista di idee su quello che l'attore può voler eseguire con il sistema (ad es.: "Login al sistema come super user") e su quello che sarebbe interessato testare (sia test positivi che negativi)
- Note sull'oracolo: come valutare il prodotto per determinare i risultati corretti (ad es. catturare cosa accade sullo schermo e confrontarlo con quello che è scritto nel manuale utente)
- Variazioni: azioni alternative e valutazioni che sono complementari alle idee descritte nelle attività.

Per gestire il testing esplorativo, può essere utilizzato un metodo chiamato *session-based test management*. Una sessione è definita come un periodo ininterrotto di testing che potrebbe durare da 60 a 120 minuti. Le sessioni di test includono:

- Sessione di sondaggio (per imparare come funziona)
- Sessione di analisi (valutazione della funzionalità o delle caratteristiche)
- Copertura di dettaglio (casi limite, scenari, interazioni).

La qualità dei test dipende dalla capacità dei tester di porre domande rilevanti su cosa testare. Alcuni esempi includono:

- Che cosa è più importante scoprire sul sistema?
- In che modo il sistema può fallire?
- Cosa succede se ...?
- Che cosa dovrebbe accadere quando ...?
- Sono soddisfatte le esigenze, i requisiti e le aspettative del cliente?
- È possibile installare il sistema (e disinstallarlo se necessario) in tutti i passi dell'aggiornamento?

Durante l'esecuzione dei test, il tester utilizza la creatività, l'intuizione, la conoscenza e la competenza per rilevare possibili problemi con il prodotto. Il tester ha anche bisogno di avere una buona conoscenza e comprensione del software sotto test, del dominio di business, di come viene utilizzato il software e di come determinare quando il sistema fallisce.

Un insieme di euristiche possono essere applicate al testing. Un'euristica può guidare il tester su come eseguire il testing e come valutare i risultati [Hendrickson]. Esempi includono:

- Limiti
- CRUD (Create, Read, Update, Delete).
- Variazioni della configurazione
- Interruzioni (ad es. log off, shut down o reboot).

È importante per il tester documentare il più possibile il processo. Altrimenti, sarebbe difficile tornare indietro e vedere come è stato rilevato un problema nel sistema. Il seguente elenco fornisce esempi di informazioni che possono essere utili per documentare:

- Copertura dei test: quali dati di input sono stati utilizzati, quanto è stato coperto e quanto resta da testare
- Note di valutazione: osservazioni durante il testing, se il sistema e la funzionalità sotto test sembrano essere stabili, se sono stati rilevati eventuali difetti, cosa è pianificato come prossimo passo in base alle attuali osservazioni, ed ogni altra lista di idee
- Elenco dei rischi/strategie: quali rischi sono stati coperti e quali rimangono tra i più importanti, se sarà seguita la strategia iniziale o se ha bisogno di eventuali modifiche
- Problemi, domande e anomalie: qualsiasi comportamento non previsto, qualsiasi domanda sull'efficienza dell'approccio, qualsiasi preoccupazione su idee/tentativi di test, sull'ambiente di test, sui dati di test, qualsiasi incomprensione sulla funzione, sugli script di test o sul sistema sotto test.
- Comportamento attuale: registrazione del comportamento attuale del sistema che deve essere salvato (ad es-, video, schermate, file di dati di output)

Le informazioni memorizzate dovrebbero essere catturate e/o sommarizzate in strumenti di status management (ad es., strumenti di test management, strumenti di task management, task board), in una modalità che renda più facile agli stakeholder comprendere lo stato attuale di tutto il testing che è stato eseguito.

3.4 Strumenti nei Progetti Agile

Gli strumenti descritti in [ISTQB_FL_SYL] sono rilevanti e utilizzati dai tester in team Agile. Non tutti gli strumenti sono utilizzati allo stesso modo e alcuni strumenti hanno più rilevanza nei progetti Agile rispetto ai progetti tradizionali. Ad esempio, anche se gli strumenti di test management, gli strumenti di requirements management e gli strumenti di incident management (strumenti di defect tracking) possono essere utilizzati dai team Agile, alcuni team Agile optano per uno strumento all-inclusive (ad es., strumenti di application lifecycle management o strumenti di task management) che fornisca funzionalità rilevanti per lo sviluppo Agile, come task board, burndown chart e user story. Gli strumenti di Configuration Management sono importanti per i tester in un team Agile a causa del numero elevato di test automatizzati a tutti i livelli e della necessità di memorizzare e gestire gli artefatti dei test automatizzati associati.

In aggiunta agli strumenti descritti in [ISTQB_FL_SYL], i tester nei progetti Agile possono utilizzare anche gli strumenti descritti nei paragrafi seguenti. Questi strumenti sono utilizzati da tutto il team per garantire la collaborazione del team e la condivisione delle informazioni, che sono la chiave delle pratiche Agile.

3.4.1 Strumenti per la Gestione e il Tracciamento dei Task

In alcuni casi, i team Agile utilizzano story board/task board fisici (ad es-, lavagne, pannelli) per gestire e tracciare le user story, i test e le altre attività nel corso di ogni sprint. Altri team utilizzeranno software di application lifecycle management e di task management, comprese task board elettroniche. Questi strumenti servono per i seguenti scopi:

- Memorizzare le user story e i task rilevanti di sviluppo e di test, per garantire che nulla venga perso durante uno sprint
- Catturare le stime dei membri del team sui loro task, e calcolare automaticamente l'effort necessario per implementare una user story, supportando sessioni efficienti di pianificazione dell'iterazione
- Associare task di sviluppo e task di test relativi alla stessa user story, per fornire un quadro completo dell'effort del team richiesto per implementare la user story
- Aggregare gli aggiornamenti dello sviluppatore e del tester allo stato del task, quando completano il loro lavoro, fornendo automaticamente una fotografia dello stato di ogni user story, dell'iterazione e della release globale
- Fornire una rappresentazione visiva (tramite metriche, grafici e dashboard), dello stato attuale di ogni user story, dell'iterazione e della release, consentendo a tutti gli stakeholder, comprese le persone in team distribuiti geograficamente, di verificare rapidamente lo stato
- Integrarsi con gli strumenti di Configuration Management, che possono consentire il tracciamento automatico del check-in del codice e delle build rispetto ai task e, in alcuni casi, aggiornare automaticamente lo stato dei task.

3.4.2 Strumenti per la Comunicazione e la Condivisione delle Informazioni

In aggiunta alle e-mail, documenti e comunicazioni verbali, i team Agile utilizzano spesso tre tipi aggiuntivi di strumenti per supportare la comunicazione e la condivisione delle informazioni: wiki, instant messaging e condivisione del desktop.

Le wiki permettono ai team di costruire e condividere una base di conoscenza online su diversi aspetti del progetto, inclusi i seguenti:

- Diagrammi delle funzionalità del prodotto, discussioni sulle funzionalità, diagrammi di prototipi, fotografie delle discussioni sulla lavagna e altre informazioni
- Strumenti e/o altre tecniche per sviluppare e testare che sono utili agli altri membri del team
- Metriche, grafici e dashboard sullo stato del prodotto, che è utile specialmente quando la wiki è integrata con altri strumenti, come il server delle build e il sistema di task management, poiché lo strumento può aggiornare lo stato del prodotto automaticamente.
- Le conversazioni tra i membri del team, similare all'instant messaging e all'e-mail, ma in un modo che sia condiviso con qualsiasi altro membro del team.

Strumenti di instant messaging, di teleconferenza audio e di video chat forniscono i seguenti benefici:

- Consentono la comunicazione in tempo reale e diretta tra i membri del team, specialmente nei team distribuiti
- Coinvolgono i team distribuiti negli standup meeting
- Riducono le bollette telefoniche utilizzando la tecnologia voice-over-IP, rimuovendo i vincoli di costo che potrebbero ridurre la comunicazione tra i membri di team distribuiti.

Gli strumenti di forniscono i seguenti vantaggi:

- Nei team distribuiti, possono essere utilizzati per demo del prodotto, per code review e anche per il pairing
- Catturano demo del prodotto alla fine di ogni iterazione, che possono essere visualizzate sulla wiki del team.

Questi strumenti dovrebbero essere utilizzati per completare ed estendere, non sostituire, la comunicazione face-to-face nei team Agile.

3.4.3 Strumenti per la Build e la Distribuzione del Software

Come discusso in precedenza in questo Syllabus, la build e la distribuzione del software giornaliera è una pratica fondamentale nei team Agile. Questo richiede l'utilizzo di strumenti di Continuous Integration e di strumenti di distribuzione della build. Gli usi, i benefici e i rischi di questi strumenti sono stati già descritti nel Par.1.2.4.

3.4.4 Strumenti di Configuration Management

In un team Agile, gli strumenti di Configuration Management possono essere utilizzati non solo per memorizzare il codice sorgente e i test automatizzati, ma spesso i test manuali e gli altri prodotti di lavoro del test sono memorizzati nello stesso repository del codice sorgente del prodotto. Questo fornisce la tracciabilità tra le versioni software che sono state testate e le particolari versioni dei test, e consente modifiche rapide senza perdere informazioni storiche. I principali tipi di sistemi di controllo delle versioni includono sistemi centralizzati di controllo del codice sorgente e sistemi distribuiti di controllo della versione. La dimensione, la struttura e la distribuzione del team, e i requisiti di integrazione con altri strumenti determineranno quale sistema di controllo delle versioni è adatto per un particolare progetto Agile.

3.4.5 Strumenti per la Progettazione, Implementazione ed Esecuzione dei Test

Alcuni strumenti sono utili ai tester Agile in momenti specifici nel processo software di test. Anche se la maggior parte di questi strumenti non sono nuovi o specifici per Agile, forniscono comunque funzionalità importanti a supporto delle modifiche veloci nei progetti Agile.

- Strumenti per la progettazione dei test: Uso di strumenti come le mappe mentali sono diventati più popolari per progettare e definire velocemente i test di una nuova funzionalità
- Strumenti di gestione dei test case: Strumenti che possono essere parte dello strumento di application lifecycle management o dello strumento di task management usato da tutto il team
- Strumenti per la preparazione e la generazione dei dati di test: Strumenti che generano dati per popolare il database di un'applicazione sono molto utili quando occorrono molti dati e combinazioni di dati per testare l'applicazione. Questi strumenti possono anche aiutare a ridefinire la struttura del database quando un prodotto subisce modifiche durante un progetto Agile e per eseguire il refactoring degli script per generare i dati. Questo permette un rapido aggiornamento dei dati di test, non appena si verificano modifiche. Alcuni strumenti di preparazione dei dati di test utilizzano i dati di produzione come base sorgente e utilizzano gli script per eliminare o anonimizzare i dati sensibili. Altri strumenti di preparazione dei dati di test possono aiutare a validare gli input e output dei dati di grandi dimensioni
- Strumenti di caricamento dei dati di test: Dopo che i dati sono stati generati per il testing, è necessario che siano caricati nell'applicazione. L'inserimento manuale dei dati spesso richiede tempo ed è soggetta ad errori, ma gli strumenti di caricamento dei dati sono disponibili per rendere il processo affidabile ed efficiente. Infatti, molti degli strumenti per la generazione dei dati includono un componente di caricamento dei dati integrato. In altri casi, è anche possibile il caricamento massivo dei dati utilizzando i sistemi di database management.

- Strumenti di esecuzione dei test automatizzati: Esistono strumenti di esecuzione dei test che sono più adatti al testing Agile. Sono disponibili sia strumenti specifici commerciali che open source, per supportare gli approcci test-first, come Behaviour-Driven Development, Test-Driven Development e Acceptance Test-Driven Development. Questi strumenti permettono ai tester e allo staff di business di esprimere il comportamento atteso del sistema in tabelle o nel linguaggio naturale utilizzando keyword.
- Strumenti di test esplorativo: Gli strumenti che catturano e memorizzano le attività eseguite su un'applicazione durante una sessione di test esplorativo sono utili per i tester e gli sviluppatori, poiché registrano le azioni eseguite. Questo è utile quando viene rilevato un difetto, poiché vengono catturate le azioni eseguite prima che si verifichi il failure, in modo che possano essere utilizzate per segnalare il difetto agli sviluppatori. La memorizzazione dei passi eseguiti in una sessione di test esplorativo può rivelarsi utile se il test è in seguito incluso nella suite dei regression test automatizzati.

3.4.6 Strumenti di Cloud Computing e di Virtualizzazione

La virtualizzazione permette a una singola risorsa fisica (server) di operare come molte risorse più piccole e separate. Quando si utilizzano virtual machine o istanze cloud, i team hanno un maggior numero di server disponibili per lo sviluppo e il testing. Questo può aiutare a evitare i ritardi associati all'attesa di server fisici. Il provisioning di un nuovo server o il ripristino di un server risulta più efficiente con le funzionalità di snapshot integrate nella maggior parte degli strumenti di virtualizzazione. Alcuni strumenti di test management ora utilizzano le tecnologie di virtualizzazione per effettuare snapshot dei server nel momento in cui viene rilevato un guasto, consentendo ai tester di condividere lo snapshot con gli sviluppatori per investigare il guasto.

4. Riferimenti

4.1 Standard

- [DO-178B] RTCA/FAA DO-178B, Software Considerations in Airborne Systems and Equipment Certification, 1992.
- [ISO25000] ISO/IEC 25000:2005, Software Engineering - Software Product Quality Requirements and Evaluation (SQuaRE), 2005.

4.2 Documenti ISTQB

- [ISTQB_ALTA_SYL] ISTQB Advanced Level Test Analyst Syllabus, Version 2012
- [ISTQB_ALTM_SYL] ISTQB Advanced Level Test Manager Syllabus, Version 2012
- [ISTQB_FA_OVIEW] ISTQB Foundation Level Agile Tester Overview, Version 1.0
- [ISTQB_FL_SYL] ISTQB Foundation Level Syllabus, Version 2011

4.3 Libri

[Aalst13] Leo van der Aalst and Cecile Davis, “TMap NEXT® in Scrum,” ICT-Books.com, 2013.

[Adzic09] Gojko Adzic, “Bridging the Communication Gap: Specification by Example and Agile Acceptance Testing,” Neuri Limited, 2009.

[Anderson13] David Anderson, “Kanban: Successful Evolutionary Change for Your Technology Business,” Blue Hole Press, 2010.

[Beck02] Kent Beck, “Test-driven Development: By Example,” Addison-Wesley Professional, 2002.

[Beck04] Kent Beck and Cynthia Andres, “Extreme Programming Explained: Embrace Change, 2e” Addison-Wesley Professional, 2004.

[Black07] Rex Black, “Pragmatic Software Testing,” John Wiley and Sons, 2007.

[Black09] Rex Black, “Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing, 3e,” Wiley, 2009.

[Chelimsky10] David Chelimsky et al, “The RSpec Book: Behavior Driven Development with Rspec, Cucumber, and Friends,” Pragmatic Bookshelf, 2010.

[Cohn04] Mike Cohn, “User Stories Applied: For Agile Software Development,” Addison-Wesley Professional, 2004.

[Crispin08] Lisa Crispin and Janet Gregory, “Agile Testing: A Practical Guide for Testers and Agile Teams,” Addison-Wesley Professional, 2008.

[Goucher09] Adam Goucher and Tim Reilly, editors, “Beautiful Testing: Leading Professionals Reveal How They Improve Software,” O'Reilly Media, 2009.

[Jeffries00] Ron Jeffries, Ann Anderson, and Chet Hendrickson, “Extreme Programming Installed,” Addison-Wesley Professional, 2000.

[Jones11] Capers Jones and Olivier Bonsignour, “The Economics of Software Quality,” Addison-Wesley Professional, 2011.

[Linz14] Tilo Linz, “Testing in Scrum: A Guide for Software Quality Assurance in the Agile World,” Rocky Nook, 2014.

[Schwaber01] Ken Schwaber and Mike Beedle, “Agile Software Development with Scrum,” Prentice Hall, 2001.

[vanVeenendaal12] Erik van Veenendaal, “The PRISMA approach”, Uitgeverij Tutein Nolthenius, 2012.

[Wiegers13] Karl Wiegers and Joy Beatty, “Software Requirements, 3e,” Microsoft Press, 2013.

4.4 Terminologia Agile

Le keyword trovate nel Glossario ISTQB sono identificate all’inizio di ogni capitolo. Per i termini comuni Agile, ci siamo basati sulle seguenti risorse Internet che forniscono le relative definizioni.

http://guide.Agilealliance.org/

http://whatis.techtargetcom/glossary

http://www.scrumalliance.org/

Incoraggiamo i lettori a verificare questi siti se trovano in questo documento termini non familiari relativi alla metodologia Agile. Questi collegamenti erano attivi al momento del rilascio di questo Syllabus.

4.5 Altri Riferimenti

I seguenti riferimenti puntano a informazioni disponibili su siti Internet. Anche se questi riferimenti sono stati verificati al momento della pubblicazione del Syllabus, ISTQB non può ritenersi responsabile se i riferimenti non sono più disponibili.

- [Agile Alliance Guide] Autori vari, http://guide.Agilealliance.org/.
- [Agilemanifesto] Autori vari, www.agilemanifesto.org.
- [Hendrickson]: Elisabeth Hendrickson, “Acceptance Test-driven Development,” testobsessed.com/2008/12/acceptance-test-driven-development-atdd-an-overview.
- [INVEST] Bill Wake, “INVEST in Good Stories, and SMART Tasks,” xp123.com/articles/invest-in-good-stories-and-smart-tasks.
- [Kubaczkowski] Greg Kubaczkowski and Rex Black, “Mission Made Possible,” www.rbcs-us.com/images/documents/Mission-Made-Possible.pdf.