Certificazione di Tester

Syllabus

Estensione del Livello ‘Foundation’

Tester nella metodologia Agile

Versione 2014

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.

Internal Reviewers: 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.

Storia delle revisioni

Versione	Data	Note
Syllabus v0.1	26/07/2013	Standalone sections
Syllabus v0.2	16/09/2013	WG review comments on v01 incorporated
Syllabus v0.3	20/10/2013	WG review comments on v02 incorporated
Syllabus v0.7	16/12/2013	Alpha review comments on v03 incorporated
Syllabus v0.71	20/12/2013	Working group updates on v07
Syllabus v0.9	30/01/2014	Beta version
Syllabus 2014	31/05/2014	GA version

Storico delle Revisioni di traduzione per ITA-STQB

REV.	AUTORI	DESCRIZIONE DELLE MODIFICHE	DATA DI APPROVAZIONE DEL COMITATO SCIENTIFICO
01	Massimo Di Carlo	Traduzione del Syllabus Estensione per il tester della Metodologia Agile	Luglio 2014

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.

0. Introduzione a questo syllabus

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

Termini:

Agile Manifesto, sviluppo software Agile, modello di sviluppo incrementale, modello di sviluppo iterativo, ciclo di vita del software, test automation, test basis, test-driven development, test oracle, 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 dello sviluppo 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 stories testabili in collaborazione con gli sviluppatori e i rappresentanti del 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 dell'integrazione continua.
FA-1.2.5 (K1) Apprendere le differenze tra la pianificazione della iterazione e della release 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 lavorerà in modo diverso di uno 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 con gli sviluppatori e i rappresentanti del business aziendale. I membri di un progetto Agile comunicano tra loro dalle prime fasi del progetto e frequentemente, il che aiuta a rimuovere i difetti il più presto possibile 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 leggero, ha concordato un insieme comune di valori e di principi che è diventato noto come il Manifesto for Agile Software Development o l'Agile Manifesto [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

Ovvero, fermo restando il valore delle voci a destra, si considerano più importanti le voci a sinistra

Individui e interazioni

Lo sviluppo Agile è molto incentrato sulle persone. Team di persone costruiscono il software ed è attraverso interazioni e comunicazioni continue, piuttosto che affidandosi a strumenti o processi, che i team possono lavorare più efficacemente.

Software funzionante

Dal punto di vista del cliente, un software funzionante è molto più utile e di valore rispetto ad una documentazione eccessivamente dettagliata e inoltre esso 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 ad un vantaggio significativo sul time-to-market. Lo sviluppo Agile è, quindi, particolarmente utile in ambienti aziendali in rapida evoluzione in cui i problemi e/o le soluzioni non sono chiare o in cui il business desidera innovare in nuovi settori di attività.

Collaborazione del Cliente

I clienti spesso incontrano molta difficoltà a specificare il sistema di cui hanno bisogno. Collaborando direttamente con il cliente si aumenta la probabilità di comprendere esattamente di cosa il cliente abbia bisogno. Mentre l'avere dei contratti con i clienti può essere importante, lavorare in collaborazione stretta e regolare con loro è probabile che conduca ad un maggiore successo del progetto.

Rispondere al cambiamento

Il cambiamento è inevitabile nei progetti software. L'ambiente in cui l'azienda opera, la legislazione, l'attività dei concorrenti, i progressi della tecnologia e altri fattori possono avere impatti importanti sul progetto ed i suoi obiettivi. Questi fattori devono essere risolti dal processo di sviluppo. In tal caso, avere flessibilità nelle pratiche di lavoro per accogliere i cambiamenti è più importante che semplicemente aderire rigidamente a un piano.

Principi

I valori chiave del Manifesto Agile sono condensati in 12 principi:

- La nostra massima priorità è soddisfare il cliente attraverso la consegna tempestiva e continua di software valido.
- I cambiamenti nei requisiti sono benvenuti, anche nelle fasi avanzate dello sviluppo. I processi Agile sfruttano il cambiamento per il vantaggio competitivo del cliente
- Rilasciare frequentemente software funzionante, ad intervalli compresi tra poche settimane a qualche mese, con una preferenza per le scadenze più brevi.
- Le persone del business e gli sviluppatori devono lavorare insieme ogni giorno 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.
- Il metodo più efficiente ed efficace di trasmettere informazioni da e all'interno di un team di sviluppo è una conversazione faccia a faccia.
- 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.
- L'essenzialità - l'arte di evidenziare la quantità di lavoro non svolto – è fondamentale.
- Le architetture, i requisiti e la progettazione migliori emergono da team che si auto-organizzano.
- Ad intervalli regolari, il team riflette su come diventare più efficace, quindi regola e adatta il suo 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 delle altre parti interessate al processo di business che determinano le caratteristiche del prodotto. Il Team dovrebbe essere relativamente piccolo; team di successo sono stati osservati con un minimo di tre persone, fino a nove persone. Idealmente, tutto il Team condivide lo stesso spazio di lavoro, perché la co-ubicazione agevola fortemente la comunicazione e l'interazione. L'approccio whole-team è supportato attraverso incontri giornalieri (vedi Paragrafo 2.2.1) che coinvolgono tutti i membri del team, in cui si comunica l'avanzamento dei lavori e sono evidenziati gli eventuali ostacoli al progresso. L'approccio whole-team promuove dinamiche di gruppo più efficaci ed efficienti.

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

- Migliorare la comunicazione e la collaborazione all'interno del team
- Far sì che le varie competenze all'interno del team siano coinvolte positivamente a vantaggio del progetto
- Rendere la qualità l'obiettivo 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 delle linee di business per garantire che i livelli di qualità desiderati siano raggiunti. Ciò comprende il sostegno e la collaborazione con i rappresentanti delle linee di business per aiutarli a creare test di accettazione adeguati, lavorare con gli sviluppatori per concordare la strategia di test e decidere sull'approccio all'automazione dei test. I tester possono così trasferire ed estendere la conoscenza del test agli altri membri del team e influenzare lo sviluppo del prodotto.

L'intero team è coinvolto in tutte le consultazioni o le riunioni in cui sono presentate, analizzate o stimate le caratteristiche del prodotto. Il concetto di coinvolgere i tester, gli sviluppatori e i rappresentanti del business in tutte le discussioni sulle funzionalità è conosciuto come il potere del tre [Crispin08]. Nel seguito del syllabus si indicherà brevemente il Team Agile con "Team"

1.1.3 Feedback precoci e frequenti

I progetti Agile hanno iterazioni brevi che consentono al team di progetto di ricevere feedback tempestivi e continui sulla qualità del prodotto durante tutto il ciclo di sviluppo. Un modo per fornire un feedback rapido è quello dell'integrazione continua (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 affinché il team di sviluppo possa affrontare efficacemente eventuali problemi che il cliente possa evidenziare. Ottenendo frequenti feedback dai clienti mentre il progetto progredisce, i team Agile possono integrare la maggior parte dei nuovi cambiamenti durante il processo di sviluppo del prodotto. Frequenti feedback sin dalle prime fasi dello sviluppo aiutano il team a concentrarsi sulle funzioni con il più elevato valore per il business o di rischio associato, che verranno perciò consegnate al cliente per prime. Ciò aiuta anche a gestire meglio il team in quanto la sua capacità di lavoro è trasparente per tutti. Ad esempio, quanto lavoro si può fare in uno sprint o iterazione? Che cosa potrebbe aiutarci ad andare più veloci? Che cosa ci impedisce di farlo?

I benefici di feedback precoci e frequenti includono:

- Evitare fraintendimenti nei requisiti, che possono non essere rilevati fino alle ultime fasi nel ciclo di sviluppo, quando sono più costosi da correggere.
- Chiarire le richieste dei clienti, rendendoli disponibili in anticipo per l'utilizzo da parte degli utenti. In questo modo, il prodotto corrisponderà di più a ciò che il cliente desidera.

- Scoprire (tramite integrazione continua), isolare e risolvere in anticipo i problemi di qualità.
- Fornire informazioni al team Agile per quanto riguarda la sua produttività e la capacità di produrre.
- Provocare un costante slancio al progetto.

1.2 Aspetti degli approcci Agile

Ci sono un certo numero di approcci Agile in uso presso le varie organizzazioni. Pratiche comuni tra la maggior parte delle organizzazioni che usano l'approccio Agile includono la creazione collaborativa delle user stories, la retrospettiva, l'integrazione continua e la pianificazione per ogni iterazione, nonché per il rilascio globale. Questo paragrafo descrive alcuni degli approcci Agile.

1.2.1 Approcci alla sviluppo software Agile

Ci sono diversi approcci Agile, ciascuno dei quali implementa i valori e i principi del Manifesto Agile in modi diversi. In questo syllabus sono considerati tre approcci Agile caratteristici: Extreme Programming (XP), Scrum e Kanban.

Extreme Programming

Extreme Programming (XP), originariamente introdotto da Kent Beck [Beck04], è un approccio Agile di sviluppo software descritto da certi 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, insuccesso, qualità, piccoli passi e responsabilità accettata.

XP descrive tredici pratiche principali: sedersi insieme, whole-team, area di lavoro informativa, lavoro sotto tensione, pair programming, storie, ciclo settimanale, ciclo trimestrale, lasco, ten–minutes build, integrazione continua, programmazione Test first e progettazione incrementale.

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

Scrum

Scrum è un'infrastruttura di gestione Agile che contiene i seguenti strumenti e pratiche [Schwaber01]:

- Sprint: Scrum divide un progetto in iterazioni (chiamate sprint) di lunghezza fissa (di solito 2-4 settimane).
- Incremento del prodotto: Ogni sprint risulta in un prodotto potenzialmente rilasciabile / installabile (chiamato incremento).
- Product Backlog: Il responsabile del prodotto gestisce un elenco prioritizzato di parti di prodotto pianificate. Il Product Backlog si evolve da sprint a sprint (chiamato raffinamento del backlog).
- Sprint Backlog: all'inizio di ogni sprint, il team Scrum seleziona un insieme di elementi a più alta priorità dal Product Backlog. Poiché il team Scrum, non il responsabile del prodotto, seleziona gli elementi da realizzare all'interno dello sprint, la selezione è detta di tipo pull piuttosto che push.
- Definizione di Done: per assicurarsi che ci sia 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 del prodotto.

- Timeboxing: Solo i task, i requisiti o le funzionalità che il team prevede di terminare all'interno dello sprint fanno parte del backlog sprint. Se il team di sviluppo non può finire un compito all'interno di uno sprint, le funzionalità associate vengono rimosse dallo sprint e il task viene spostato indietro 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 una riunione).
- Trasparenza: Il team di sviluppo fa una sintesi dell'avanzamento sullo stato dello sprint e lo aggiorna su base giornaliera in una riunione chiamata il daily scrum. Questo rende visibile il contenuto e il progresso dello sprint corrente, compresi i risultati dei test, al team, al management e a tutte le parti interessate. Ad esempio, il team di sviluppo può mostrare lo stato dello sprint su una lavagna.

Scrum definisce tre ruoli:

- Scrum Master: assicura che le pratiche e le regole siano implementate e rispettate e risolve ogni violazione, problemi di risorse o altri impedimenti che potrebbero impedire al team di seguire le pratiche e le regole. Questa persona non è il capo del team, ma piuttosto colui che facilita una corretta esecuzione del processo.
- Product Owner: rappresenta il cliente e genera, mantiene e dà le priorità al Product Backlog. Questa persona non è il team leader.
- Team di sviluppo: sviluppa e testa il prodotto. Il team è auto-organizzato: Non vi è alcun team leader, quindi il Team prende le decisioni. Il Team è anche cross-funzionale (si veda il paragrafo 2.3.2 e il paragrafo 3.1.4).

Scrum (rispetto a XP) non detta specifiche tecniche di sviluppo software (per es. programmazione del tipo test first). In aggiunta, Scrum non fornisce una guida su come deve essere fatto il test nel progetto Scrum.

Kanban

Kanban è un approccio di gestione che è usato alcune volte nei progetti Agile. L'obiettivo generale è visualizzare e ottimizzare il flusso di lavorazione all'interno della catena di valore aggiunto. Kanban utilizza tre strumenti [Linz14]:

- Kanban Board: La catena di valore da gestire è visualizzata da una Kanban Board. Ciascuna colonna mostra una stazione, che è un insieme di attività relative, per es. sviluppo o collaudo. Gli elementi che devono essere prodotti o i task che devono essere eseguiti sono simbolizzati da biglietti che si muovono da sinistra a destra sulla lavagna attraverso le stazioni.
- Work-in-Progress Limit: il numero di task paralleli attivi è strettamente limitato. Ciò è controllato dal massimo numero di biglietti che possono stare su una stazione e/o globalmente sulla lavagna. Ogni volta che una stazione ha capacità libera, il lavoratore prende un biglietto dalla precedente stazione.
- Lead Time: Kanban è usato per ottimizzare il flusso continuo di task minimizzando il tempo (medio) di attraversamento per l'intero flusso di valore.

Le caratteristiche di Kanban sono simili a quelle di Scrum. In entrambe le infrastrutture, visualizzare i task attivi (per es. su una lavagna visibile a tutti) fornisce trasparenza al contenuto e al progresso dei task. I task non ancora schedulati sono in attesa in un backlog e si muovono in avanti sulla Kanban board appena c'è nuovo spazio (capacità produttiva) disponibile,

Iterazioni o sprint sono opzionali in Kanban. Il processo Kanban consente di rilasciare i suoi prodotti elemento per elemento, piuttosto che come parte di una release. Il Timeboxing come meccanismo di sincronizzazione, quindi, è opzionale, diversamente da Scrum che 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, caratteristiche ridondanti o contraddittorie e altre difficoltà di comunicazione degli utenti. Nello sviluppo Agile, le user story sono scritte per catturare i requisiti dal punto di vista degli sviluppatori, tester e rappresentanti del business. Nello sviluppo sequenziale, questa visione condivisa di una caratteristica si realizza attraverso

revisioni formali dopo che sono stati scritti i requisiti; nello sviluppo agile, questa visione condivisa si realizza attraverso frequenti revisioni informali mentre vengono scritti i requisiti.

Le user stories devono affrontare sia le caratteristiche funzionali che non funzionali. Ogni storia include i criteri di accettazione per queste caratteristiche. Tali criteri dovrebbero essere definiti in collaborazione tra rappresentanti del business, sviluppatori e tester. Essi forniscono agli sviluppatori e ai tester una visione estesa della funzione che i rappresentanti del business valideranno. Un Team Agile considera terminato un task quando una serie di criteri di accettazione sono stati soddisfatti.

Tipicamente, la sola prospettiva del tester migliorerà la user story identificando dettagli mancanti o requisiti non funzionali. Un tester può contribuire ponendo ai rappresentanti del business domande aperte sulla user story, proponendo modi per testare la storia utente 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
- **N**egoziabile
- **V**alutabile
- Stimabile (***E****stimable*)
- Piccolo (***S****mall*)
- **T**estabile

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

- Card: La card è il supporto fisico che descrive una storia utente. Esso identifica il requisito, la sua criticità, durate attese dello sviluppo e del test, ed i criteri di accettazione per quella storia. La descrizione deve essere accurata, poiché verrà utilizzata nel product backlog.
- Conversazione: 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 i rappresentanti del business [ISTQB_FL_SYL], portano un valido contributo allo scambio di idee, opinioni ed esperienze. La conversazione inizia durante la fase di pianificazione del rilascio e continua quando la user story è in fase di esecuzione.
- Conferma: I criteri di accettazione, discussi nella conversazione, sono utilizzati per confermare che la user story è realizzata. Questi criteri di accettazione possono estendersi su più user stories. Sia i test positivi che negativi devono essere utilizzati per coprire i criteri. Durante la conferma, vari partecipanti svolgono il ruolo di un tester. Questi possono comprendere tanto gli sviluppatori quanto specialisti focalizzati su prestazioni, sicurezza, interoperabilità e altre caratteristiche di qualità. Per confermare una storia come completata, i criteri di accettazione definiti dovrebbero essere testati per provare che siano stati soddisfatti.

I Team Agile variano in termini di come documentano le user story. Indipendentemente dall'approccio adottato per documentare le user stories, la documentazione deve essere concisa, sufficiente e necessaria.

1.2.3 Retrospettive

Nello sviluppo Agile, una retrospettiva è una riunione tenuta alla fine di ciascuna iterazione per discutere su cosa ha avuto successo, cosa potrebbe essere migliorato e come incorporare i miglioramenti e mantenere cosa ha avuto successo nelle iterazioni future. Le retrospettive coprono argomenti come il processo, le persone, le organizzazioni, le relazioni e i tools. Incontri di retrospettiva tenuti regolarmente, quando ci sono appropriate attività di controllo, sono critiche per l'auto organizzazione e il miglioramento continuo dello sviluppo e del testing.

Le retrospettive possono portare a decisioni di miglioramento collegate al test centrate sulla efficacia e la produttività del test, la qualità dei casi di test e la soddisfazione del Team. Possono anche indirizzare la testabilità delle applicazioni, le user stories, le funzioni o le interfacce del sistema.

La Root cause analysis dei difetti può portare a miglioramenti nel test e nello sviluppo. In generale, i Team Agile devono implementare solo pochi miglioramenti per iterazione. Ciò consente un miglioramento continuo ad un passo sostenibile.

Il timing e l'organizzazione della retrospettiva dipendono dalla particolare metodologia Agile seguita. I rappresentanti del business e il Team partecipano ad ogni retrospettiva come partecipanti mentre il facilitatore organizza e guida l'incontro. In alcuni casi, i Teams possono invitare altri partecipanti alla riunione.

I tester dovrebbero svolgere un ruolo importante nelle retrospettive. I tester sono parte del Team e portano il loro punto di vista particolare ISTQB_FL_SYL], Paragrafo 1.5. Il test viene eseguito in ogni sprint e contribuisce in modo determinante al successo. Tutti i membri del Team, tester e non, possono fornire un contributo sia alle attività di test che 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 revisione come discusso nel Syllabus del livello Foundation [ISTQB_FL_SYL], Paragrafo 3.2.

1.2.4 Integrazione Continua

La realizzazione di un incremento del prodotto richiede che sia realizzato un software integrato, funzionante, affidabile alla fine di ogni sprint. La integrazione continua indirizza questa sfida fondendo tutti i cambiamenti fatti al software e integrando tutti i componenti in modo regolare almeno una volta al giorno. La gestione della configurazione, la compilazione , la build del software, l'installazione e il test sono raccolte in un singolo processo automatizzato e ripetibile. Poiché gli sviluppatori in modo costante integrano il loro lavoro, ne effettuano la build e lo testano, i difetti nel codice sono scoperti più velocemente.

Utilizzando un repository del codice condiviso da cui prendere il codice per la codifica e il debugging, un processo di integrazione continua consiste nelle seguenti attività automatizzate:

- Analisi statica del codice: esecuzione dell'analisi statica del codice e reporting dei risultati
- Compilazione: compilare e linkare il codice, generando i file eseguibili
- Unit test: esecuzione degli unit tests, controllando la copertura del codice e effettuando il reporting dei risultati
- Deploy: installazione della build nell'ambiente di test
- Test di integrazione: esecuzione dei test di integrazione e reporting dei risultati
- Report (cruscotto): inviare lo stato di tutte queste attività in un posto visibile in modo condiviso o inviarlo via e-mail al Team.

Un processo di build e test automatico ha luogo su base giornaliera e scopre errori di integrazione prima e velocemente. L'integrazione continua permette ai tester Agile di eseguire test automatizzati regolarmente, in alcuni casi come parte dello stesso processo di integrazione continua e manda un rapido feedback al Team sulla qualità del codice. Questi risultati del test sono visibili a tutti i membri del Team, specialmente quando report automatici sono integrati nel processo. Test di regressione automatizzati possono essere continui attraverso tutta l'iterazione. Buoni test di regressione coprono quante più funzionalità è possibile, incluse le user stories rilasciate nelle precedenti iterazioni.

Una buona copertura nei test di regressione automatizzati aiuta a supportare la build (e il collaudo) di grandi sistemi integrati. Quando il test di regressione è automatizzato, i tester Agile sono liberi di concentrare l'esecuzione dei test manuali sulle nuove funzioni, sui cambiamenti implementati e i test di conferma delle anomalie risolte.

In aggiunta ai test automatizzati, le strutture che usano l'integrazione continua, tipicamente usano strumenti di build per implementare controlli di qualità continui. 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 controllo della qualità. Questa continua applicazione dei controlli della qualità punta a migliorare la

qualità del prodotto come a ridurre il tempo impiegato per rilasciare il prodotto stesso rimpiazzando la pratica tradizionale di applicare i controlli di qualità dopo il completamento di tutto lo sviluppo.

Gli strumenti di build possono essere collegati a strumenti per l'installazione automatica, che possono estrarre la build corretta dal server di integrazione continua o di build e installarla in uno o più ambienti di sviluppo, test, staging o anche di produzione. Ciò riduce gli errori e i ritardi associati all'affidarsi a gruppi specializzati o a programmatori per installare le release in tali ambienti.

L'integrazione continua può dare i seguenti benefici:

- Permette di scoprire in anticipo ed eseguire più facilmente la root cause analysis di problemi di integrazione e modifiche in conflitto tra loro
- Dà al Team di sviluppo un feedback regolare sul fatto che il codice è funzionante
- Fa sì che la versione sia testata entro un giorno da quando è stata sviluppata
- Riduce il rischio di regressione associato alle modifiche del software grazie al rapido re-test dopo ogni piccolo insieme di modifiche
- Dà confidenza che ogni lavoro di sviluppo giornaliero sia basato su solide fondamenta
- Rende visibili i progressi rispetto al completamento del prodotto, incoraggiando gli sviluppatori e i tester
- Elimina i rischi di pianificazione associati all'integrazione di tipo big-bang
- Fornisce costante disponibilità di software eseguibile in tutto lo sprint per test, presentazioni o per istruzione
- Riduce le attività ripetitive di test manuale
- Fornisce un veloce feedback sulle decisioni prese per migliorare la qualità e i test

Comunque l'integrazione continua non è priva di rischi e sfide:

- Devono essere introdotti e manutenuti strumenti per l'integrazione continua
- Il processo di integrazione continua deve essere definito e avviato
- L'automazione dei test richiede risorse aggiuntive e può essere complessa da avviare
- Una copertura di test accurata è essenziale per ottenere i vantaggi dei test automatizzati
- I Teams a volte fanno un eccessivo affidamento sugli unit test ed eseguono troppo pochi test di sistema e di accettazione

L'integrazione continua richiede l'uso di strumenti, inclusi gli strumenti per il test, per automatizzare il processo di build e per il controllo delle versioni.

1.2.5 Pianificazione della release e dell'iterazione

Come menzionato nel syllabus Foundation Level [ISTQB_FL_SYL], la pianificazione è un'attività continua e ciò vale anche nel caso del ciclo di vita Agile. Per i cicli di vita Agile, si presentano due tipi di pianificazione: della release e dell'iterazione.

La pianificazione della release traguarda il rilascio di un prodotto, spesso un paio di mesi di anticipo rispetto l'inizio di un progetto. La pianificazione della release definisce e ri-definisce il Product Backlog e può portare ad affinare le user story più grandi in un insieme di user story più piccole. La pianificazione della release fornisce la base per un approccio al test e un piano di test che coprono tutte le iterazioni. Piani di release sono ad alto livello.

Nella pianificazione della release, rappresentanti del business stabiliscono le priorità delle user story per la release, in collaborazione con il Team (cfr. paragrafo 1.2.2). Sulla base di queste user story, sono identificati i rischi di progetto e di qualità e viene eseguita una stima ad alto livello dell'effort (cfr. Paragrafo 3.2).

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

- Definizione delle user stories testabili, inclusi i criteri di accettazione.
- Partecipazione all'analisi dei rischi di progetto e di qualità
- Stima dello sforzo per il test associato alle user stories.
- Definizione dei livelli di test necessari.
- Pianificazione del test per la release.

Dopo che è stata fatta la pianificazione della release, parte la pianificazione dell'iterazione per la prima iterazione. La pianificazione dell'iterazione prevede le attività fino alla fine di una singola iterazione e costituisce lo sprint backlog.

Nella pianificazione dell'iterazione, il Team seleziona le user stories secondo le priorità del backlog di release, elabora le user stories, esegue l'analisi del rischio per le user stories e ne stima il lavoro necessario per ciascuna. Se una user story è troppo vaga e i tentativi di chiarirla sono falliti, il Team può rifiutare di accettarla e usa la prossima user story in base alla priorità. I rappresentanti del business devono rispondere alle domande del Team sulla user story in modo che il Team possa comprendere meglio ciò che deve realizzare e come testare ciascuna storia.

Il numero di user stories selezionate è basato sulla velocità del Team e la dimensione stimata delle user stories selezionate. Dopo che siano stati definiti i contenuti dell'iterazione, le user stories sono divise in task, che saranno svolti dai membri del Team.

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

- Partecipazione nell'analisi dettagliata del rischio delle user stories.
- Determinazione della testabilità delle user stories.
- Creazione dei test di accettazione per le user stories.
- Divisione delle user stories in task (in particolare task di test).
- Stima dello sforzo del test per tutti i task di test.
- Identificazione degli aspetti funzionali e non funzionali del sistema da testare.
- Supporto e partecipazione all'automazione del test ai vari livelli di test.

I piani di release possono cambiare con il procedere del progetto, comprese le modifiche a singole user stories nel Product Backlog. Questi cambiamenti possono essere innescati da fattori interni o esterni. I fattori interni comprendono capacità di erogazione, velocità e questioni tecniche. I fattori esterni includono la scoperta di nuovi mercati e opportunità, nuovi concorrenti o problemi di business che possono cambiare gli obiettivi della release e/o le scadenze. Inoltre, i piani di iterazione possono cambiare durante l'iterazione. Ad esempio, una particolare user story che è stata considerata relativamente semplice durante la stima potrebbe rivelarsi più complessa del previsto.

Questi cambiamenti possono essere sfidanti per i tester. I tester devono capire il quadro generale della release ai fini della pianificazione del test e devono avere una base di test adeguata e un test oracle in ogni iterazione per gli obiettivi di test di sviluppo come descritto nel syllabus a livello Foundation [ISTQB_FL_SYL], Paragrafo 1.4. Le informazioni richieste devono essere disponibili al tester presto, ma il cambiamento deve essere accolto secondo i principi Agile. Questa questione richiede attente decisioni sulle strategie e la documentazione di test. Per maggiori informazioni sulle sfide del test Agile, vedere [Black09], Capitolo 12.

La pianificazione della release e delle iterazioni deve indirizzare la pianificazione del test tanto quanto la pianificazione delle attività di sviluppo. Questioni particolari da risolvere relativamente al test includono:

- Lo scopo del test, l'estensione del test per le aree in ambito, gli obiettivi del test e le ragioni di queste decisioni.
- I membri del Team che eseguiranno le attività di test.
- L'ambiente di test, i dati di test necessari, quando sono necessari e se ogni aggiunta o cambiamento all'ambiente e/o ai dati di test avverrà prima o durante il progetto.
- I tempi, la sequenza, le dipendenze e i prerequisiti per le attività di test funzionali e non funzionali (per es. quanto frequentemente eseguire i test di regressione, quali funzioni dipendono da altre funzioni o dai dati di test, ecc.) incluso come le attività di test sono collegate e dipendono dalle attività di sviluppo.
- I rischi di progetto e di qualità che devono essere affrontati.

In aggiunta, la stima dello sforzo per i Team più grandi deve prendere in considerazione il tempo e l'impegno necessario per completare le attività di test richieste.

2. Principi, regole e processi fondamentali del test Agile – 105 minuti

Termini:

Build verification test, configuration item, configuration management.

Obiettivi di apprendimento per Principi, regole e processi fondamentali del test Agile.

2.1 Le differenze tra il test nell'approccio tradizionale e in quello Agile

FA-2.1.1 (K2) Descrivere le differenze tra le attività di test in progetti Agile e progetti non-Agile
FA-2.1.2 (K2) Descrivere come le attività di sviluppo e test sono integrate nei progetti Agile.
FA-2.1.3 (K2) Descrivere il ruolo del test indipendente nei progetti Agile.

2.2 Stato del test nei progetti Agile

FA-2.2.1 (K2) Descrivere i tools e le tecniche usati per comunicare lo stato del test in un progetto Agile, inclusi i progressi del test e la qualità del prodotto
FA-2.2.2 (K2) Descrivere il processo di evoluzione dei test attraverso iterazioni multiple e spiegare perché l'automazione del test è importante pe gestire il rischio di regressione nei progetti Agile.

2.2 Ruolo e skills di un tester in un Team Agile

FA-2.3.1 (K2) Comprendere gli skills (persona, dominio di conoscenza e test) 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 test nell'approccio tradizionale e in quello Agile

Come descritto nel syllabus Foundation Level [ISTQB_FL_SYL] e in [Black09], le attività di test sono legate alle attività di sviluppo e quindi il test varia a seconda dei diversi cicli di vita. I tester devono capire le differenze tra il test nei modelli del ciclo di vita tradizionali (ad esempio, sequenziali come il V-model o iterativi come RUP) e cicli di vita di tipo Agile, al fine di lavorare in modo efficace ed efficiente. I modelli Agile differiscono nei termini di come le attività di test e di sviluppo sono integrate, i prodotti del lavoro del progetto, i nomi, i criteri di entrata e di uscita utilizzati per i vari livelli di test, l'uso di strumenti e come il test indipendente può essere efficacemente utilizzato.

I tester dovrebbero ricordare che le organizzazioni variano notevolmente nella loro implementazione del ciclo di vita del software. Una deviazione dagli ideali del ciclo di vita Agile (vedi il paragrafo 1.1) può rappresentare una personalizzazione ed un adattamento intelligente delle regole. 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, ogni iterazione che conduce ad un software funzionante che offre caratteristiche di valore per gli stakeholder aziendali. All'inizio del progetto, vi è un periodo di pianificazione della release. Questo è seguito da una sequenza di iterazioni. All'inizio di ogni iterazione, vi è un periodo di pianificazione dell'iterazione. Una volta stabiliti gli obiettivi dell'iterazione, le user stories selezionale sono sviluppate, integrate con il sistema e testate. Queste iterazioni sono altamente dinamiche, con attività di sviluppo, integrazione e test che si svolgono durante ogni iterazione e con notevole

parallelismo e sovrapposizioni. Le attività di testing si effettuano durante tutta l'iterazione, non come attività finale.

Tester, sviluppatori e i referenti del business hanno tutti un ruolo in fase di test, come nei cicli di vita tradizionali. Gli sviluppatori eseguono gli unit test quando sviluppano le funzioni dalle user stories. I tester poi testano tali funzioni. Anche i referenti del business testano le user stories durante l'implementazione. i referenti del business potrebbero usare casi di test scritti, ma anche potrebbero semplicemente eseguire un test esplorativo e utilizzare la funzione al fine di fornire un feedback veloce per il team di sviluppo.

In alcuni casi, si svolgono periodicamente iterazioni di consolidamento o di stabilizzazione per risolvere eventuali difetti persistenti e altre forme di *technical debt*. Tuttavia, la pratica migliore è che nessuna funzione è considerata fatta fino a quando non sia stata integrata e testata con il sistema [Goucher09]. Un'altra buona pratica è quella di risolvere i difetti rimanenti dalla precedente iterazione all'inizio della successiva iterazione, come parte del backlog per tale iterazione (di seguito "fissare prima i bugs"). Tuttavia, alcuni lamentano che questa pratica porta ad una situazione in cui il lavoro totale che deve essere fatto nella iterazione è sconosciuto e sarà più difficile stimare quando le restanti funzioni possono essere realizzate. Alla fine della sequenza di iterazioni, ci può essere una serie di attività di rilascio per ottenere il software pronto per la consegna, anche se in alcuni casi la consegna avviene al termine di ogni iterazione.

Quando il testing basato sul rischio è usato come una delle strategie di test, un'analisi dei rischi di alto livello viene effettuata durante la pianificazione della release, con i tester che spesso guidano tale analisi. Tuttavia, i rischi specifici di qualità associati a ogni iterazione vengono individuati e valutati nella pianificazione dell'iterazione. Questa analisi del rischio può influenzare la sequenza di sviluppo, nonché la priorità e la profondità del test delle funzioni. Essa influenza anche la stima dell'effort del test richiesto per ciascuna funzione (vedi Paragrafo 3.2).

In alcune pratiche Agile (ad esempio, Extreme Programming), viene utilizzato il lavoro in coppia, il che prevede che due tester lavorino insieme per testare una funzione. Oppure può comportare che un tester lavori in collaborazione con uno sviluppatore per sviluppare e testare una funzione. Il lavoro in coppia può essere difficile quando il team di test è distribuito, ma processi e strumenti possono contribuire a permettere il lavoro in coppia distribuito. Per ulteriori informazioni sul lavoro distribuito, vedere [ISTQB_ALTM_SYL], Paragrafo 2.8.

I tester possono anche servire da motivatori e coordinatori per il test e la qualità all'interno del Team, condividendo le conoscenze sul test e sostenendo il lavoro per assicurare la qualità all'interno del Team. Questo promuove un senso di proprietà collettiva della qualità del prodotto.

L'Automazione dei test a tutti i livelli di test si attua in molti Team Agile e questo può significare che i tester trascorrono del tempo per la creazione, l'esecuzione, il monitoraggio e il mantenimento dei test automatizzati e dei risultati. A causa del pesante uso dell'automazione dei test, una percentuale più alta di test manuali nei progetti Agile tende ad essere fatta utilizzando tecniche basate sull'esperienza e sulla base di difetti, quali attacchi di software, test esplorativi ed error guessing (cfr. [ISTQB_ALTA_SYL], Paragrafi 3.3 e 3.4 e [ISTQB_FL_SYL], Paragrafo 4.5). Mentre gli sviluppatori si concentreranno sulla creazione di unit test, i tester dovrebbero concentrarsi sulla creazione dei test automatizzati di integrazione, di sistema e di system integration. Questo porta ad una tendenza per i Team Agile a favorire i tester con un forte background tecnico e di automazione dei test.

Un principio Agile fondamentale è che il cambiamento può verificarsi nel corso del progetto. Pertanto, nei progetti Agile è preferita una documentazione di prodotto leggera. Modifiche alle funzionalità esistenti hanno implicazioni sul test, in particolare implicazioni sul test di regressione. L'uso di test automatizzati è un modo per gestire la quantità di sforzo del test associato con il cambiamento. Tuttavia, è importante che il tasso di cambiamento non superi la capacità del team di progetto di affrontare i rischi associati a tali cambiamenti.

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 deve essere realizzato (per es. specifiche dei requisiti) e come usarlo (per es. documentazione utente).
2. Prodotti di lavoro di sviluppo che descrivono come il sistema è costruito (per es. diagrammi entità – relazioni delle basi dati), ciò che effettivamente implementa il sistema (per es. il codice) o che valuta singoli pezzi di codice (per es. unit test automatizzati).
3. Prodotti di lavoro del test che descrivono come il sistema è testato (per es: strategia di test e piano di test), ciò che effettivamente testa il sistema (per es. test manuali e automatizzati) o che presenta i risultati del test (per es. cruscotti di test come discusso nel paragrafo 2.2.1).

In un tipico progetto Agile, è pratica comune evitare di produrre una gran mole di documentazione. Invece, ci si concentra di più sull'avere un software funzionante, insieme con test automatizzati che dimostrino la conformità con i requisiti. Questo invito a ridurre la documentazione si applica solo alla documentazione che non porta valore per il cliente. In un progetto Agile di successo, si deve bilanciare tra l'aumento dell'efficienza ottenuto riducendo la documentazione e la necessità di fornire una documentazione sufficiente per supportare il business, il test, lo sviluppo e le attività di manutenzione. Il Team deve decidere durante la pianificazione della release quale documentazione è richiesta ed a quale livello.

Tipici prodotti di lavoro orientati al business nei progetti Agile includono le user stories e i criteri di accettazione. Le user stories sono il formato Agile delle specifiche dei requisiti e devono spiegare come il sistema si deve comportare rispetto ad una singola caratteristica o funzione. Una user story deve definire una funzione sufficientemente piccola da essere completata in una singola iterazione. Insiemi più grandi di funzioni collegate o un insieme di sotto-funzioni che realizzano un singola funzione complessa, possono essere riferite come "epics". Le *epics* possono includere user stories per diversi Team di sviluppo. Per esempio, una user story può descrivere quello che è richiesto a livello delle API (middleware) mentre un'altra story descrive quello che serve a livello di GUI (applicazione). Questi insiemi possono essere sviluppati in una serie di sprints. Ogni *epic* e le sue user stories devono essere associate a criteri di accettazione.

Un tipico prodotto di lavoro per lo sviluppatore nei progetti Agile è il codice. Gli sviluppatori Agile spesso creano anche unit test automatizzati. Questi test possono essere creati dopo lo sviluppo del codice. In alcuni casi, tuttavia, gli sviluppatori creano i test in modo incrementale, prima che sia scritta ciascuna porzione di codice, per fornire un modo per verificare, una volta che la porzione di codice sia scritta, se funziona come previsto. Mentre questo approccio è indicato come test first o sviluppo basato su test, in realtà i test sono più una forma di specifiche di progetto di basso livello eseguibili piuttosto che test [Beck02].

Tipici prodotti di lavoro per i tester nei progetti Agile includono i test automatizzati, così come i documenti quali i piani di test, cataloghi dei rischi di qualità, test manuali, report dei difetti e logs dei risultati dei test. I documenti vengono acquisiti in modo più leggero possibile, il che spesso è anche vero per questi documenti nei cicli di vita tradizionali. I tester produrranno anche metriche dei test dai report sui difetti e sui risultati dei test e di nuovo si pone l'enfasi su un approccio leggero.

In alcune implementazioni Agile, soggetti a particolari regole, safety critical, distribuiti o progetti e prodotti altamente complessi, è necessaria un'ulteriore formalizzazione di questi prodotti di lavoro. Ad esempio, alcuni Team trasformano user stories e criteri di accettazione in specifiche dei requisiti più formali. Report di tracciabilità verticali e orizzontali possono essere preparati per soddisfare i revisori, i regolamenti 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 della componente 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, specifiche di progettazione e le attività di sviluppo possono sovrapporsi con i livelli di test.

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

- Unit test, tipicamente fatto dallo sviluppatore.
- Test di accettazione della funzione, che talvolta è diviso in due attività:
    - Test di verifica della funzione, che è spesso automatizzato, può essere eseguito dagli sviluppatori o dai tester e comporta il test dei criteri di accettazione della user story.
    - Test di validazione della funzione, che è di solito manuale e può coinvolgere gli sviluppatori, i tester e i rappresentanti del business che lavorano insieme per determinare se la funzione è corretta dal punto di vista dell'utilizzo previsto, per aumentare la visibilità dei progressi fatti e per ricevere un reale riscontro dai rappresentanti del business.

Inoltre, vi è spesso un processo parallelo di test di regressione che avviene durante l'iterazione. Si tratta di ri-esecuzione degli unit test automatizzati e test di verifica delle funzioni della iterazione corrente e di quelle precedenti, di solito tramite un framework di integrazione continua.

In alcuni progetti Agile, ci può essere un livello di test di sistema, che inizia una volta che la prima user story è pronta per tali test. Questo può comportare l'esecuzione di test funzionali, come non funzionali per prestazioni, affidabilità, usabilità e altri tipi di test pertinenti.

I Team Agile possono utilizzare varie forme di test di accettazione (usando il termine come spiegato nel syllabus Foundation Level [ISTQB_FL_SYL]). Possono essere eseguiti alfa test interni e beta test esterni, alla fine di ogni iterazione, dopo il completamento di tutte le iterazioni o dopo una serie di iterazioni. Possono anche essere eseguiti i test di accettazione utente, test di accettazione operativi, test di accettazione per le normative e test di accettazione di contratto, sia alla fine di ogni iterazione, dopo il completamento di ogni iterazione o dopo una serie di iterazioni.

2.1.4 Test e Configuration Management

I progetti Agile spesso comportano un uso pesante di strumenti automatizzati per sviluppare, testare e gestire lo sviluppo del software. Gli sviluppatori utilizzano strumenti di analisi statica, di unit test e di copertura del codice. Gli sviluppatori gestiscono il codice e lo unit test tramite un sistema di gestione della configurazione, eseguendo build automatizzate e framework di test. Questi framework consentono l'integrazione continua del nuovo software con il sistema, con l'analisi statica e gli unit test eseguiti ripetutamente ogni qualvolta il nuovo software viene caricato nel sistema di gestione della configurazione [Kubaczkowski].

Questi test automatizzati possono includere anche test funzionali ai livelli di integrazione e di sistema. Tali test funzionali automatizzati possono essere creati utilizzando ambienti di test funzionali, strumenti commerciali o open-source per i test funzionali tramite interfaccia utente e possono essere integrati con i test automatizzati eseguiti come parte del framework di integrazione continua. 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 che un nuovo software viene sviluppato, mentre i test funzionali, per cui serve più tempo, vengono eseguiti solo ogni tanto.

Uno degli obiettivi dei test automatizzati è di confermare che la build è funzionante e installabile. Se un test automatizzato fallisce, il Team dovrebbe risolvere il difetto in tempo per il successivo rilascio del codice. Ciò richiede un investimento nel reporting tempestivo dei test per fornire una buona visibilità dei loro risultati. Questo approccio consente di ridurre i cicli costosi e inefficienti di "build-install-fail-rebuild-reinstall" che possono verificarsi in molti progetti tradizionali, dal momento che i cambiamenti che fanno fallire la build o provocano errori nell'installazione del software vengono rilevati rapidamente.

Gli strumenti per l'automazione dei test e delle build aiutano a gestire il rischio di regressione associato ai frequenti cambiamenti che si verificano spesso nei progetti Agile. Tuttavia, l'eccessiva dipendenza solo sugli unit test automatizzati per gestire questi rischi può essere un problema, poiché

lo unit test spesso ha un'efficacia limitata nel rilevare difetti [Jones11]. Sono necessari anche test automatizzati a livello di integrazione e di sistema.

2.1.5 Scelte organizzative per il test indipendente

Come discusso nel syllabus Foundation Level [ISTQB_FL_SYL], i tester indipendenti sono spesso più efficaci nel trovare i difetti. In alcuni Team Agile, gli sviluppatori creano molti dei test in forma 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, vi è un rischio di perdita di indipendenza e di valutazione non obiettiva.

Altri Team Agile mantengono team di test separati e completamente indipendenti, e assegnano i tester a richiesta durante i giorni finali di ogni sprint. Questo può preservare l'indipendenza e questi tester possono fornire una valutazione obiettiva ed imparziale del software. Tuttavia le pressioni sui tempi, la limitata comprensione delle nuove funzionalità del prodotto e problemi di relazione con gli stakeholder aziendali e gli sviluppatori portano spesso a problemi anche 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 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 ad esempio lo sviluppo di strumenti di test automatizzati, l'esecuzione di test non funzionali, la creazione la e gestione di ambienti di test e dati ed l'esecuzione di livelli di test che potrebbero non adattarsi bene all'interno di uno sprint (ad esempio, test di integrazione di sistema).

2.2 Stato del test nei progetti Agile

Una modifica viene gestita rapidamente nei progetti Agile. Una modifica implica che lo stato del test, il suo avanzamento e la qualità del prodotto evolvono costantemente e i tester devono trovare il modo per trasmettere tali informazioni al Team, in modo che si possano prendere le decisioni necessarie per rispettare i tempi previsti per ilcompletamento dell'iterazione. Inoltre, la modifica può influenzare le funzioni già realizzate nelle iterazioni precedenti. Pertanto, test manuali e automatici devono essere aggiornati per affrontare efficacemente il rischio di regressione.

2.2.1 Comunicare lo stato del test, gli avanzamenti e la qualità del prodotto

I Team Agile operano producendo software funzionante alla fine di ogni iterazione. Per determinare quando il Team avrà del software funzionante, è necessario monitorare l'avanzamento di tutti i work items nell'iterazione e nella release. I tester nei Team Agile utilizzano vari metodi per registrare l'avanzamento e lo stato del test, compresi i risultati dell'automazione dei test, l'avanzamento delle attività di test e le user story, sulla Agile task board e sui grafici burndown che mostrano i progressi del Team. Questi possono poi essere comunicati al resto del Team utilizzando supporti quali cruscotti wiki, e-mail in stile cruscotto, nonché verbalmente nel corso di riunioni sullo stato di avanzamento. I Team Agile possono utilizzare strumenti che generano automaticamente i report di stato, basati sui risultati dei test e sullo stato di avanzamento dell'attività, che a loro volta aggiorneranno i cruscotti 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 casi di test.

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 che resta da fare rispetto al tempo previsto per la release o l'iterazione

Per fornire una rappresentazione visiva istantanea e dettagliata dello stato attuale di tutto il Team, compreso lo stato del test, i Team possono utilizzare le Agile task boards. Le story card, le attività di

sviluppo, le attività di test e le altre attività create durante la pianificazione dell'iterazione (cfr. paragrafo 1.2.5) vengono rappresentate sulla task board, spesso utilizzando carte di colore differente per determinare il tipo di attività. Durante l'iterazione, l'avanzamento è gestito tramite lo spostamento di queste attività attraverso la task board suddivisa in colonne come *cose da fare*, *attività in corso*, *da verificare* e *fatto*. I Team Agile per mantenere le story card e le Agile task board, possono utilizzare strumenti che possono automatizzare i cruscotti e gli aggiornamenti di stato.

Le attività di test sulla task board riguardano i criteri di accettazione definiti per le user stories. Quando gli script di automazione dei test, i test manuali e i test esplorativi raggiungono un passaggio di stato, l'attività si sposta nella colonna *fatto* della task board. L'intero Team esamina lo stato della task board regolarmente, spesso durante gli incontri giornalieri sullo stato di avanzamento, per accertare che le attività si muovano attraverso la task board ad una velocità accettabile. Se alcune attività (comprese le attività di test) non si muovono o si muovono troppo lentamente, il Team esamina e affronta eventuali problemi che possono bloccare il progresso di tali attività.

Le riunioni di avanzamento giornaliere coinvolgono tutti i membri del Team Agile, inclusi i tester, che comunicano lo stato corrente delle loro attività. L'agenda per ogni membro è [Agile Alliance Guide]:

- Cosa hai completato dall'ultima riunione
- Cosa hai pianificato di completare entro la prossima riunione
- Quali ostacoli stai incontrando

Ogni problema che possa bloccare l'avanzamento del test è comunicato durante le riunioni di avanzamento giornaliero, in modo che tutto il Team sia consapevole dei problemi e di conseguenza li possa risolvere.

Per migliorare la qualità complessiva del prodotto, molti Team Agile eseguono indagini di customer satisfaction per ricevere feedback se il prodotto soddisfa le aspettative dei clienti. I Team possono utilizzare altri parametri simili a quelli ottenuti in metodologie di sviluppo tradizionali, come il rapporto tra test falliti e passati, i tassi di scoperta dei difetti, i risultati dei test di regressione e di conferma, la densità dei difetti, il numero di difetti scoperti e risolti, la copertura dei requisiti, la copertura dei rischi, la copertura del codice e la varianza del codice. Come con qualsiasi ciclo di vita, le metriche acquisite e comunicate devono essere pertinenti ed 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 regressione evolvendo i casi di test manuali e automatizzati

In un progetto Agile, come completamento di ogni iterazione, il prodotto cresce. Pertanto, l'ambito del test aumenta. Oltre a testare le modifiche al codice apportate nell'iterazione corrente, i tester devono anche verificare che nessuna regressione sia stata introdotta sulle funzioni che sono state sviluppate e testate nelle iterazioni precedenti. Il rischio di introdurre regressioni nello sviluppo Agile è elevato a causa di una vasta dinamica del codice (righe di codice aggiunte, modificate o cancellate da una versione all'altra). Dal momento che la risposta al cambiamento è un principio chiave dell'Agile, le modifiche possono essere fatte anche su funzionalità consegnate in precedenza per soddisfare esigenze di business. Per mantenere la velocità senza incorrere in una grande quantità di debito tecnico, è fondamentale che i Team investano nell'automazione dei test a tutti i livelli il più presto possibile. É anche fondamentale che tutti gli oggetti del test, come i test automatizzati, i casi di test manuali, i dati di test e altri prodotti di test , siano tenuti aggiornati ad ogni iterazione. E' altamente raccomandato che tutti i test siano inseriti in uno strumento di gestione della configurazione per consentire il controllo di versione, per garantire facilità di accesso da parte di tutti i membri del Team e per supportare le modifiche richieste da cambiamenti alle funzionalità, pur conservando le informazioni storiche di tutti gli oggetti del test.

Poiché la riesecuzione completa di tutti i test è possibile solo raramente, specialmente in progetti Agile con scadenze strette, i tester devono dedicare del tempo in ogni iterazione per rivedere i casi di test manuali e automatici delle iterazioni precedenti (e della iterazione corrente) allo scopo di selezionare i casi di test candidabili ad una suite di test di regressione e di eliminare i casi di test non

più utili. I test scritti nelle prime iterazioni per verificare specifiche funzionalità possono avere poco valore nelle successive iterazioni, poiché funzionalità nuove o modificate possono alterare il comportamento di tali funzionalità. .

Durante la revisione dei casi di test, i tester ne dovrebbero prendere in considerazione l'idoneità per l'automazione. Il Team ha bisogno di automatizzare il maggior numero di test possibili dalle iterazioni precedenti e da quella attuale. Ciò permette di eseguire i test di regressione in modo automatizzato riducendo il rischio di regressione con uno sforzo minore rispetto a quello che richiederebbero i test di regressione manuali. Questo ridotto sforzo di test di regressione permette di testare più a fondo le nuove caratteristiche e le nuove funzionalità nell'iterazione corrente.

E' fondamentale che i tester abbiano la capacità di identificare rapidamente ed aggiornare i casi di test delle iterazioni e/o release precedenti, che sono impattati dalle modifiche apportate nella iterazione corrente. Durante la pianificazione della release si deve definire come il Team progetta, scrive e conserva i casi di test. Buone pratiche per la progettazione e l'implementazione dei test devono essere adottate sin dall'inizio ed applicate in modo coerente. La riduzione dei tempi per i test e le frequenti modifiche in ogni iterazione aumenteranno invece l'impatto negativo di una progettazione ed una implementazione di scarsa qualità.

L'uso dell'automazione dei test, 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]. Inserendo i test automatizzati ed i loro corrispondenti risultati nel sistema di gestione della configurazione, ed allineandoli con il versioning delle build del prodotto, i Team Agile possono rivedere le funzionalità testate ed i risultati del test per ogni build in un qualsiasi momento.

Gli unit test automatizzati vengono eseguiti prima che il codice sorgente venga archiviato nella mainline del sistema di gestione della configurazione, per garantire che le modifiche al codice non corrompano la build del software. Per ridurre le anomalie di compilazione delle build, che possono rallentare il progresso di tutto il Team, il codice non dovrebbe essere archiviato 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.

Test di accettazione automatizzati vengono eseguiti regolarmente come parte dell'integrazione continua della build completa del sistema. Questi test vengono eseguiti su una build del sistema completo almeno ogni giorno, ma non sono in genere eseguiti ogni volta che una parte del software viene archiviata nel sistema di gestione della configurazione, poiché richiedono più tempo dell'esecuzione degli unit test automatizzati e potrebbero rallentare l'archiviazione del codice. I risultati dei test di accettazione automatizzati forniscono un feedback sulla qualità del prodotto rispetto alla regressione dopo l'ultima build, ma non forniscono lo stato della qualità complessiva del prodotto.

I test automatizzati possono essere eseguiti continuamente sul sistema. Un sottoinsieme iniziale di test automatizzati per coprire le funzionalità del sistema e dei punti di integrazione dovrebbe essere creato immediatamente dopo che una nuova build venga installata nell'ambiente di test. Questi test sono comunemente noti come test di verifica della build. I risultati dei test di verifica della build forniranno un feedback immediato sul software dopo l'installazione, per cui i Team non perdono tempo a testare una build instabile.

I test automatizzati contenuti nell'insieme dei test di regressione sono in genere eseguiti, come parte della build principale giornaliera, nell'ambiente di integrazione continua e anche quando una nuova build viene installata nell'ambiente di test. Non appena un test di regressione automatizzato fallisce, il Team si ferma e indaga le ragioni del fallimento del test. Il test potrebbe essere fallito a causa di cambiamenti funzionali previsti nella iterazione corrente, nel qual caso il test e/o la user story possono avere bisogno di essere aggiornati per riflettere i nuovi criteri di accettazione. In alternativa, il test può essere eliminato se un altro test è stato realizzato per coprire le modifiche. Tuttavia, se il test è fallito a causa di un difetto, è una buona pratica che il Team risolva il difetto prima di progredire con nuove funzionalità.

Oltre all'automazione dei test, 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 (per es. il database o i file dati di un sito web)
- Comparazione dei dati di output

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

2.3 Ruolo e skills di un tester in un Team Agile

In un Team Agile, i tester devono collaborare strettamente con tutti gli altri membri del Team e con i rappresentanti del business. Ciò 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 Skills di un Tester Agile

I tester Agile dovrebbero avere tutte le competenze menzionate nel syllabus Foundation Level [ISTQB_FL_SYL]. In aggiunta a queste competenze, un tester in un Team Agile deve essere competente nell'automazione dei test, nello sviluppo test-driven, nello sviluppo basato su test di accettazione, nei test di tipo white-box, black-box e nel test basato sull'esperienza.

Poiché le metodologie Agile dipendono fortemente dalla collaborazione, dalla comunicazione e dall'interazione tra i membri del Team e gli altri soggetti 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 sia con i membri del Team che gli attori coinvolti
- Mostrare un atteggiamento critico, orientato alla qualità e scettico verso il prodotto
- Acquisire attivamente informazioni dagli stakeholders (piuttosto che affidarsi completamente alle specifiche scritte)
- Valutare accuratamente i risultati e l'avanzamento del test e la qualità del prodotto e effettuarne i relativi report
- Lavorare in modo efficace, con i rappresentanti dei clienti e le altre parti interessate, per definire user stories testabili, in particolare i criteri di accettazione
- Collaborare con il Team, lavorare in coppia con i programmatori e gli altri membri del Team
- Rispondere alle modifiche rapidamente, eventualmente modificando, aggiungendo o migliorando i casi di test
- Pianificare e organizzare il proprio lavoro.

La crescita continua delle competenze, tra cui la crescita delle capacità relazionali è essenziale per tutti i tester, inclusi quelli dei Team Agile.

2.3.2 Il ruolo di un Tester in un Team Agile

Il ruolo di un tester in un Team Agile comprende attività che generano e forniscono feedback non solo sullo stato e l'avanzamento del test e la qualità del prodotto, ma anche sulla qualità del processo. Oltre alle attività descritte in altre parti di questo Syllabus, tali attività comprendono:

- Capire, implementare e aggiornare la strategia di test
- Misurare e comunicare la copertura del test rispetto a tutti i criteri di copertura applicabili
- Garantire il corretto utilizzo 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
- Dare supporto agli altri membri del Team in aspetti rilevanti del test
- Garantire che appropriate attività di test siano previste durante la pianificazione della release e dell'iterazione
- Collaborare attivamente con gli sviluppatori ed i clienti per chiarire i requisiti, soprattutto in termini di testabilità, di coerenza e di completezza
- Partecipare proattivamente alle riunioni di retrospettiva del Team, suggerendo e implementando dei miglioramenti

All'interno di un Team Agile, ogni membro del Team è responsabile della qualità del prodotto e svolge un ruolo nello svolgimento di compiti collegati al test.

Organizzazioni di tipo Agile possono incontrare alcuni rischi organizzativi collegati al test:

- I tester, lavorando a così stretto contatto con gli sviluppatori, perdono il giusto atteggiamento mentale di tester
- I tester diventano tolleranti o silenti sulle pratiche inefficienti, inefficaci o di bassa qualità all'interno del Team
- I tester non possono tenere il passo con i cambiamenti in arrivo in iterazioni con scadenze 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 al test, criteri di accettazione, dichiarazione di test, esecuzione automatizzata del test, framework di unit test, rischio di prodotto, rischio di qualità, stima del test, strategia di test, Test Driven Development, test esplorativo, test prestazionale, test di regressione.

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 dello Test Driven Development, dell'Acceptance Test Driven Development e dello behaviour driven development
FA-3.1.2 (K1) Richiamare i concetti della piramide del test
FA-3.1.3 (K2) Riassumere i quadranti del test e le loro relazioni con i livelli di test e i tipi di test
FA-3.1.4 (K3) Per un dato progetto Agile, esercitarsi sul ruolo del tester in un Team Scrum

3.2 Valutare i rischi di qualità e stimare lo sforzo del test

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 test 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 casi di test per l'Acceptance Test Driven Development
FA-3.3.4 (K3) Sia per il comportamento funzionale che non funzionale, scrivere casi di test usando le tecniche di progettazione del test black box a partire dalle user stories.
FA-3.3.4 (K3) Eseguire il test esplorativo per supportare il test di un progetto Agile

3.4 Strumenti nei progetti 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

Ci sono alcune pratiche di test che possono essere seguite in ogni progetto di sviluppo (Agile o meno) per produrre prodotti di qualità. Queste includono la scrittura di test in anticipo per verificare un comportamento correttodel prodotto, il concentrarsi sulla precoce prevenzione, individuazione e rimozione del difetto ed il garantire che i corretti tipi di test siano eseguiti al momento giusto e come parte del corretto livello di test. I professionisti Agile mirano a introdurre queste pratiche quanto prima. I tester nei progetti Agile svolgono un ruolo chiave nello stimolare 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

Il Test Driven Development, L'Acceptance Test Driven Development e il behaviour driven development sono tre tecniche complementari in uso tra i Team Agile per eseguire i test tra i vari livelli di test. Ogni tecnica è un esempio di un principio fondamentale del test, il vantaggio di effettuare quanto prima le attività di test e di QA, in quanto i test sono definiti prima che il codice sia scritto.

Test Driven Development

Il Test Driven Development è usato per sviluppare codice guidato da casi di test automatizzati. Il processo per il Test Driven Development comprende:

- Aggiungere un test che registra l'idea del programmatore sul funzionamento desiderato di un piccolo pezzo di codice.
- Eseguire subito tale test, che deve fallire in quanto il codice non esiste ancora.
- Scrivere il codice ed eseguire il test a stretto giro fino a quando il test non abbia esito positivo.
- Rivedere il codice dopo che il test è superato, rieseguire il test per assicurare 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 che quelli aggiunti.

I test scritti sono in primo luogo a livello di unit test e sono centrati sul codice, sebbene i test possono anche essere scritti a livello di integrazione o di sistema. Il Test Driven Development ha ottenuto la sua popolarità attraverso l'Extreme Programming [Beck02], ma viene utilizzato anche in altre metodologie Agile e talvolta in cicli di vita sequenziali. Aiuta gli sviluppatori a concentrarsi su risultati attesi chiaramente definiti. I test sono automatizzati e sono utilizzati nell'integrazione continua.

Acceptance Test Driven Development

L'Acceptance Test Driven Development definisce i criteri di accettazione e i test durante la creazione delle user stories (si veda il paragrafo 1.2.2). L'Acceptance Test Driven Development è un approccio collaborativo che consente ad ogni stakeholder di capire come il componente software debba comportarsi, di cosa gli sviluppatori, i tester ed i clienti hanno bisogno per garantire tale comportamento. Il processo dell'Acceptance Test Driven Development è spiegato nel paragrfo 3.3.2.

L'Acceptance Test Driven Development crea test riutilizzabili per il test di regressione. Strumenti specifici supportano la creazione e l'esecuzione di tali test, spesso nell'ambito del processo di integrazione continua. Questi strumenti possono connettersi ai layer di dati e di servizio dell'applicazione, il che permette di eseguire test a livello di sistema o di accettazione. L'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

Il behaviour driven development [Chelimsky10] permette allo sviluppatore di concentrarsi sul test del codice in base al comportamento previsto del software. Poiché i test sono basati sul comportamento esibito dal software, i test sono generalmente più facili da capire per gli altri membri del Team e per gli altri stakeholder.

Specifiche infrastrutture per il behaviour driven development possono essere usate per definire i criteri di accettazione basati sul format *dato/quando/allora*:

dato il contesto iniziale,

quando avviene un certo evento,

allora verificare uno specifico risultato.

Da tali requisiti, il framework di behaviour driven development genera codice che può essere utilizzato dagli sviluppatori per creare casi di test. Il behaviour driven development aiuta lo

sviluppatore a collaborare con le altre parti interessate, compresi i tester, per definire unit test accurati focalizzati su esigenze di business.

3.1.2 La piramide del 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 (vedi [ISTQB_FL_SYL], Paragrafo 2.2). La piramide del test sottolinea che si avranno un gran numero di test ai livelli più bassi (base della piramide) e, come le attività progettuali si spostano 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 sono automatizzati e sono creati utilizzando strumenti basati su API. A livello di sistema e di accettazione, i test automatici vengono creati utilizzando strumenti basati sulle GUI. Il concetto di piramide del test si basa sul principio di QA e testing anticipati (cioè, eliminando i difetti il più presto possibile nel ciclo di vita).

3.1.3 Quadranti del test, livelli di test e tipi di test

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

Nei quadranti di test, i test possono essere rivolti verso il business (utente) o la 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, o manuali ma supportati da strumenti. I quattro quadranti sono i seguenti:

- Quadrante Q1: è a livello di componente, rivolto alla tecnologia e supporta gli sviluppatori. Questo quadrante contiene gli unit test. Questi test dovrebbero essere automatizzati e inclusi nel processo di integrazione continua.
- Quadrante Q2: è a livello di sistema, rivolto al business e verifica il comportamento del prodotto. Questo quadrante contiene i test funzionali, esempi, test di user story, prototipi di user experience e simulazioni. Questi test controllano i criteri di accettazione e possono essere manuali o automatici. Essi sono spesso creati durante lo sviluppo della user story e quindi migliorano la qualità delle user stories stesse. Sono utili per la creazione di suite di test di regressione automatizzati.
- Quadrante Q3: è il livello di sistema o di accettazione utente, rivolto al business e contiene i test che verificano funzionalmente il prodotto, utilizzando scenari e dati realistici. Questo quadrante contiene test esplorativi, scenari, flussi di processo, test di usabilità, test di accettazione utente, alpha test e beta test. Questi test sono spesso manuali e sono user-oriented.
- Quadrante Q4: è il livello di sistema o di accettazione operativo, rivolto alla tecnologia e contiene test che verificano tecnicamente il prodotto. Questo quadrante contiene test prestazionali, di carico, di stress e scalabilità, i test di sicurezza, di manutenzione, di gestione della memoria, di compatibilità e di interoperabilità, di migrazione dei dati, di infrastruttura e i test di recovery. Questi test sono spesso automatizzati.

Durante ogni iterazione, possono essere richiesti i test di alcuni o di tutti i quadranti. I quadranti di test si applicano ai test dinamici piuttosto che ai test statici.

3.1.4 Il ruolo del tester

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

Lavoro di squadra

Il lavoro di squadra è un principio fondamentale per lo sviluppo Agile. L'Agile sottolinea l'approccio whole team composto da sviluppatori, tester e rappresentanti del business che lavorano insieme. Di seguito sono riportate le migliori pratiche organizzative e comportamentali in un Team Scrum:

- Cross-functional: Ogni membro del Team porta un diverso insieme di competenze. Il Team lavora insieme alla strategia di test, alla pianificazione dei test, alla specifica e all'esecuzione dei test, alla valutazione dei test, ed al reporting dei risultati.
- Auto organizzazione: 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.
- Co-abitazione: i tester lavorano fisicamente insieme agli sviluppatori ed ai clienti
- Collaborazione: i tester collaborano con gli altri membri del Team, con gli altri Team, gli altri stakeholders, i clienti e lo Scrum Master
- Empowered: le decisioni tecniche sulla progettazione ed il test sono prese dall'intero Team (sviluppatori, tester e Scrum Master), in collaborazione con il cliente e gli altri Team se necessario.
- Committed: il tester è impegnato a 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 test è visibile sulla task board Agile (si veda il paragrafo 2.2.1).
- Credibile: Il tester deve garantire la credibilità della strategia per il test, la sua attuazione e l'esecuzione, altrimenti gli stakeholder non si fidano dei risultati del test. Questo è spesso fatto fornendo informazioni agli stakeholder sul processo di test.
- Apertura 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.
- Resilienza: il test deve essere in grado di rispondere ai cambiamenti, come tutte le altre attività nei progetti Agile.

Queste best practices massimizzano la probabilità di eseguire un test 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 il paragrafo 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 progettare dei prototipi di alto livello
- Pianificare, acquistare e installare gli strumenti necessari (per es. per la gestione del test, la gestione dei difetti, l'automazione dei test e l'integrazione continua
- Creare una strategia di test iniziale per tutti i livelli di test, indirizzando (fra gli altri aspetti) l'ambito del test, i rischi tecnici, i tipi di test (si veda il paragrafo 3.1.3) e gli obiettivi di copertura
- Eseguire una analisi iniziale del rischio di qualità (si veda il paragrafo 3.2.1)
- Definire le metriche del test per misurare il processo di test, l'avanzamento del test nel progetto e la qualità del prodotto
- Specificare la definizione di "done"
- Creare la task board (si veda paragrafo 2.2.1)

- Definire quando continuare o fermare il test prima di rilasciare il sistema al cliente.

Lo sprint zero indica gli obiettivi del test e come li deve raggiungere attraverso i vari sprint.

Integrazione

Nei progetti Agile, l'obiettivo è quello di fornire valore al cliente su base continuativa (preferibilmente in ogni sprint). A tal fine, la strategia di integrazione deve considerare sia la progettazione che il test. 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 del test

Poiché il test è completamente integrato nel Team Agile, la pianificazione dei test dovrebbe iniziare durante la sessione di pianificazione della release e sarà aggiornata durante ogni sprint. La pianificazione dei test per la release e per ogni sprint deve affrontare i problemi discussi nella sezione 1.2.5.

La pianificazione dello Sprint si concretizza in una serie di attività da mettere sulla task board, dove ogni attività dovrebbe avere una lunghezza di uno o due giorni di lavoro. Inoltre, eventuali problemi di test dovrebbero essere monitorati per mantenere un flusso costante di test.

Pratiche di test Agile

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

- Lavoro in coppia: Due membri del Team (ad es. un tester e uno sviluppatore, due tester o un tester e il cliente) siedono insieme davanti ad una workstation per eseguire un test o altri task dello sprint.
- Progettazione del test incrementale: i casi di test e le chart sono implementati gradualmente dalle user stories e da altre basi del test, a partire dai test semplici e andando verso quelli più complessi.
- Mappe mentali: le mappe mentali sono uno strumento utile per il test [Crispin08]. Per esempio, i tester possono usare le mappe mentali per identificare quali sessioni di test eseguire, per mostrare 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 del Syllabus a livello Foundation [ISTQB_FL_SYL].

3.2 Valutare i rischi di qualità e stimare lo sforzo 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 lo sforzo necessario per ridurre a sufficienza tali rischi e quindi limitare tali rischi attraverso la progettazione, l'implementazione e l'esecuzione dei test. Tuttavia, date le brevi iterazioni e il tasso di variazione dei progetti Agile, sono necessari alcuni adattamenti di queste tecniche.

3.2.1 Valutare i rischi di qualità nei progetti Agile

Una delle tante sfide nei test è la corretta selezione, allocazione e prioritizzazione delle condizioni di test. Questo comprende la determinazione della quantità di sforzo adeguato da allocare in modo da coprire ciascuna condizione di test e mettere in sequenza i risultanti dei test in modo da ottimizzare l'efficacia e l'efficienza del lavoro di test che resta da fare. Le strategie di identificazione dei rischi, di analisi e di mitigazione del rischio possono essere usate dai tester in un Team Agile per aiutare a determinare un numero accettabile di casi di test da eseguire, anche se molti vincoli e variabili che interagiscono possono richiedere dei compromessi.

Il rischio è la possibilità che accada un risultato o un evento negativo o non atteso. Il livello di rischio è trovato valutando la probabilità di accadimento del rischio ed il suo impatto. Quando l'effetto primario del potenziale problema è sulla 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 i rischi di progetto o rischi di pianificazione [Black07] [vanVeenendaal12].

Nei progetti Agile, l'analisi del rischio di qualità ha luogo in due occasioni:

- Nella pianificazione della release: i rappresentanti del business che conoscono le funzioni della release forniscono una vista ad alto livello dei rischi ed il resto del Team, inclusi i tester, può aiutare nella identificazione e nella valutazione del rischio.
- Nella pianificazione dell'iterazione: il Team nel suo complesso identifica e valuta i rischi di qualità.

Esempi di rischi di qualità per un sistema sono:

- Calcoli errati nei report (un rischio funzionale legato all'accuratezza).
- Risposta lenta agli input dell'utente (un rischio non funzionale legato alla efficienza e al tempo di risposta).
- Difficoltà di comprendere schermate e campi (rischio non funzionale legato all'usabilità e alla comprensibilità).

Come accennato in precedenza, una iterazione inizia con la sua pianificazione, che culmina in attività stimate sulla task board. Queste attività possono essere parzialmente prioritizzate in base al livello di rischio di qualità ad esse associato. Attività connesse con rischi più elevati dovrebbero iniziare prima e coinvolgere più attività di test. Attività connesse con rischi minori dovrebbero iniziare più tardi e richiedere meno sforzo di test.

Un esempio di come il processo di analisi del rischio di qualità in un progetto Agile possa essere effettuato durante la pianificazione dell'iterazione è delineato nei seguenti passaggi:

1. Mantenere i membri del Team Agile insieme, inclusi i tester.
2. Elencare tutte le voci del backlog per la iterazione corrente (per es. sulla task board).
3. Identificare i rischi di qualità associati con ciascuna voce, considerando tutti gli aspetti rilevanti per la qualità.
4. Valutare ogni rischio identificato, il che comporta due attività: assegnare una categoria al rischio e determinarne il livello di rischio basato sull'impatto e la probabilità di occorrenza dei difetti.
5. Determinare l'estensione del test proporzionale al livello di rischio.
6. Selezionare le appropriate tecniche di test per mitigare ciascun rischio, in base al tipo di rischio, il livello di rischio e gli aspetti rilevanti per la qualità.

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

Nel corso del progetto, il Team deve sempre recepire ulteriori informazioni che possano cambiare l'insieme dei rischi e/o il livello di rischio associato ai rischi di qualità già noti. Il Team deve anche effettuare un adeguamento periodico dell'analisi dei rischi di qualità, che si traduce in adeguamenti dei test. Le modifiche comprendono l'individuazione di nuovi rischi, la ri-valutazione 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 del test. Ad esempio, se sono trovati problemi con le user stories durante l'identificazione del rischio, il team di progetto può rivedere con attenzione le user stories come strategia di attenuazione.

3.2.2 Stima del test basata sui contenuti e sul rischio

Durante la pianificazione della release, il Team Agile stima lo sforzo richiesto per completare la release. La stima riguarda anche lo sforzo per il test. 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. Ognuno di questi ha un mazzo di carte con valori simili alla sequenza di Fibonacci (cioè, 0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...) o qualsiasi altra progressione di scelta (ad esempio, dimensioni delle camicie da extra-small a extra-extra-large). I valori rappresentano il numero di story points, di sforzo in giorni o altre unità in cui il Team effettua la stima. La sequenza di Fibonacci è consigliata perché i numeri nella sequenza riflettono il fatto che l'incertezza cresce proporzionalmente con la dimensione della storia. Una stima alta di solito significa che la storia non è ben compresa o dovrebbe essere suddivisa in storie più piccole.

Le persone che devono effettuare la stima discutono la funzione e pongono domande al Product Owner, se necessario. Aspetti come lo sviluppo e le attività di test, la complessità della storia e la portata dei test 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 ampiamente discussa, ciascuno stimatore sceglie, senza mostrarla agli altri, una cartaper indicare la sua stima. Tutte le carte sono 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 norme (ad esempio, utilizzando la mediana, utilizzando il punteggio più alto) per limitare il numero di giri di poker. Queste discussioni assicurano una stima attendibile dello sforzo 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, devono essere fatte alcune considerazioni specifiche e delle variazioni nelle tecniche di test, nella terminologia e nella documentazione.

3.3.1 Criteri di accettazione, copertura adeguata e altre informazioni sul test

I progetti Agile delineano requisiti iniziali come user stories in un backlog prioritizzato all'inizio del progetto. I requisiti iniziali sono brevi e di solito seguono un formato predefinito (cfr. paragrafo 1.2.2). Sono anche importanti i requisiti non funzionali, quali l'usabilità e le prestazioni, e possono essere specificati come user stories uniche o collegate ad altre user stories funzionali. I requisiti non funzionali possono seguire un formato predefinito o standard, come [ISO25000], oppure una norma specifica del settore.

Le user stories servono come un'importante base del test. Altre importanti basi per il test includono:

- Esperienze da progetti precedenti.
- Funzioni, caratteristiche o aspetti di qualità già esistenti nel sistema.
- Codice, architettura e documenti di 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 taxonomia dei difetti.
- Standard applicabili (per es. [DO-178B] per il software avionico).

- Rischi di qualità (si veda il paragrafo 3.2.1).

Durante ogni iterazione, gli sviluppatori realizzano codice che implementa le funzioni e le caratteristiche descritte nelle user stories, con le caratteristiche di qualità necessarie e questo codice viene verificato e convalidato tramite test di accettazione. Per essere testabili, i criteri di accettazione devono rispettare i seguenti aspetti, ove applicabili [Wiegers13]:

- Comportamento funzionale: il comportamento osservabile dall'esterno con l'operatività dell'utente come alcuni dati di input con date configurazioni.
- Caratteristiche di qualità: come il sistema esegue il comportamento specificato. Le caratteristiche possono anche far riferimento ad attributi di qualità o a requisiti non funzionali. Caratteristiche di qualità tipiche sono le prestazioni, l'affidabilità, l'usabilità, ecc.
- Scenari (casi d'uso): una sequenza di azioni fra un attore esterno (spesso un utente) e il sistema per eseguire un compito specifico o un'attività di business.
- Regole di business: attività che possono solo essere eseguite nel sistema sotto certe condizioni definite da procedure esterne e vincoli (per es. le procedure usate dalle compagnie di assicurazione per gestire reclami).
- Interfacce esterne: descrizioni delle connessioni fra il sistema da sviluppare e il mondo esterno. Le interfacce esterne possono essere divise in tipi differenti (interfacce utente, interfacce con altri sistemi, ecc.).
- Vincoli: ogni vincolo di progettazione o di implementazione che restringe le opzioni per lo sviluppatore. I dispositivi con software embedded spesso devono rispettare vincoli fisici quali le dimensioni, il peso e le connessioni di interfaccia.
- Definizione dei dati: Il cliente può descrivere il formato, il tipo di dati, i valori consentiti e i valori di default per un dato nella composizione di una struttura di dati complessa (per esempio, il codice di avviamento postale in un indirizzo).

In aggiunta alle user stories ed i loro corrispondenti criteri di accettazione, sono importanti altre informazioni per il test, quali:

- Come il sistema deve funzionare ed essere utilizzato.
- Le interfacce di sistema che possono essere utilizzate / accedute per testare il sistema.
- Se il supporto allo strumento utilizzato è sufficiente.
- Se il tester ha abbastanza conoscenze e competenze per effettuare i test necessari.

I tester spesso scoprono la necessità di ulteriori informazioni (per es. la copertura di codice) durante tutte le iterazioni e devono lavorare in collaborazione con il resto dei membri del Team Agile per ottenere tali informazioni. Le informazioni pertinenti svolgono un ruolo nel determinare se una particolare attività può essere considerata *done*. Questo concetto della definizione di *done* è critica in progetti Agile e si applica in un numero di modi diversi come discusso nei seguenti sottoparagrafi.

Livelli di test

Ciascun livello di test ha la propria definizione di *done.* La seguente lista dà degli esempi che possono essere importanti per i differenti livelli di test:

- Unit test
    - 100% della copertura delle decisioni dove possibile, con revisioni accurate di ogni cammino non possibile
    - Esecuzione dell'analisi statica su tutto il codice
    - Nessun difetto importante non risolto (classificato in base alla priorità e alla severità)
    - Nessun debito tecnico conosciuto e non accettabile rimasto nella progettazione e nel codice [Jones11]

- Completata la revisione di tutto il codice, degli unit test e dei risultati degli unit test
- Tutti gli unit test automatizzati
- Le caratteristiche importanti sono entro il limiti previsti (per es. le prestazioni)

- Test di integrazione
  - Tutti i requisiti funzionali testati, compresi i test positivi e negativi, con il numero di test calcolato in base alle dimensioni, complessità e rischi.
  - Tutte le interfacce fra i componenti testate.
  - Tutti i rischi di qualità coperti in accordo con la copertura concordata del test.
  - Nessun difetto importante non risolto (classificato in base alla priorità e all'importanza)
  - E' stato fatto un record completo di tutti i difetti scoperti
  - Ove possibile, tutti i test di regressione sono stati automatizzati e memorizzati in un repository comune.
- Test di sistema
  - Effettuati i test end-to-end delle user stories, le caratteristiche e le funzioni.
  - Sono stati coperti tutti i possibili profili utente
  - Le più importanti caratteristiche di qualità del sistema sono state coperte (per es. prestazioni, robustezza, affidabilità).
  - Eseguito il test in un ambiente simile a quello di produzione, compreso tutto l'hardware e il software per tutte le configurazioni supportate per quanto possibile.
  - Tutti i rischi di qualità coperti in accordo con la copertura concordata del test.
  - Ove possibile, tutti i test di regressione sono stati automatizzati e memorizzati in un repository comune.
  - Tutti i difetti scoperti sono stati comunicati e se possibile risolti.
  - Non rimane da risolvere alcun difetto importante (classificato in base al rischio e all'importanza)

User Story

La definizione di *done* per le user stories può essere determinato dai seguenti criteri:

- Le user stories selezionate per l'iterazione sono complete, comprese dal Team e hanno criteri di accettazione dettagliati e testabili.
- Tutti gli elementi della user story sono specificati e rivisti, inclusi i test di accettazione della user story.
- I task necessari per implementare e testare le user stories sono stati identificati e stimati dal Team.

Funzionalità

La definizione di *done* per le funzionalità, che possono estendersi a più storie utente o epics, possono comprendere:

- Tutte le user stories componenti, con i relativi criteri di accettazione, sono definite e approvate dal cliente.
- La progettazione è completa, con nessun debito tecnico.
- Il codice è completo, con nessun debito tecnico o modifiche non completate.
- Sono stati eseguiti gli unit test e sono stati ottenuti i livelli di copertura previsti

- I Test di integrazione e di sistema per le funzionalità sono stati eseguiti secondo i definiti criteri di copertura.
- Nessun difetto principale deve essere ancora corretto.
- La documentazione delle funzionalità è completa, incluse le release note, i manuali utenti e le funzioni di help on line.

Iterazioni

La definizione di *done* per le iterazioni può comprendere:

- Tutte le funzioni dell'iterazione sono pronte e testate singolarmente secondo i criteri a livello di funzione
- Ogni difetto non critico, che non può essere risolto entro i confini dell'iterazione, *è* stato aggiunto al Product Backlog e prioritizzato.
- L'integrazione di tutte le funzionalità dell'iterazione è completata e testata.
- La documentazione è scritta, rivista 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 definizione di *done* per una release*,* che può estendersi a più iterazioni, può comprendere le seguenti aree:

- Copertura: tutti gli elementi della base di test rilevanti per tutti i contenuti della release sono state coperti dai test. L'adeguatezza della copertura è determinata da ciò che è nuovo o modificato, dalla sua complessità e dalle dimensioni ed i rischi associati di fallimento.
- Qualità: l'intensità dei difetti (ad esempio, quanti difetti sono trovati al giorno o per transazione), la densità dei difetti (ad esempio, il numero di difetti riscontrati rispetto al numero di user stories, lo sforzo, e/o gli attributi di qualità), il numero stimato dei difetti rimanenti +è entro limiti accettabili, le conseguenze dei difetti irrisolti e rimanenti (ad esempio, la severità e la priorità) sono compresi ed accettabili, il livello residuo di rischio associato a ciascun rischio di qualità individuato è compreso ed accettato.
- Tempo: se la data prevista di consegna è stata raggiunta, devono essere considerati gli impatti sul business associati al rilascio o al mancato rilascio del software.
- Costi: I costi stimati del ciclo di vita devono essere utilizzati per calcolare il ritorno di investimento per il sistema rilasciato (cioè, i costi calcolati di sviluppo e manutenzione dovrebbe essere notevolmente inferiore rispetto alle vendite totali attese del prodotto). La parte principale del costo del ciclo di vita viene spesso dalla manutenzione dopo che il prodotto è stato rilasciato, a causa del numero di difetti non trovati.

3.3.2 Applicazione dell'Acceptance Test Driven Development

L'Acceptance Test Driven Development è un approccio test-first. I casi di test sono creati prima di implementare la user story. I casi di test vengono creati dal Team Agile, tra cui lo sviluppatore, il tester, ed i rappresentanti del business [Adzic09] e possono essere manuali o automatici. Il primo passo è una riunione specifica in cui la user story viene analizzata, discussa e scritta da sviluppatori, tester e rappresentanti del business. Qualsiasi incompletezza, ambiguità o errori nella user story sono risolti durante questo processo.

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

Tipicamente i primi test sono i test positivi che confermano il corretto funzionamento senza eccezioni o condizioni di errore, compresa la sequenza delle attività eseguite se tutto va come atteso. Dopo che sono stati eseguiti tutti i cammini di test positivi, il Team deve scrivere i test per i cammini negativi e coprire anche gli attributi non funzionali (per es. prestazioni, usabilità). I test sono espressi in un modo che ogni stakeholder li possa capire e conterranno frasi in 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 devono aggiungere nulla alla user story. Ciò significa che non deve esistere un esempio che descriva un aspetto della user story non documentato nella storia stessa. Inoltre, non ci devono essere due esempi che descrivano le stesse caratteristiche della storia utente.

3.3.3 Progettazione del test black box funzionale e non funzionale

Nel test Agile, molti test sono creati dai tester in concomitanza con le attività di programmazione degli sviluppatori. Proprio come gli sviluppatori programmano in base alle user stories ed i criteri di accettazione, così anche i tester creano test basati sulle user stories ed i 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 paragrafo 3.3.4.) I tester possono applicare tecniche tradizionali di progettazione black box dei test come le classi di equivalenza, l'analisi del valore limite , le tabelle decisionali e il test di transizione di stato. Per esempio, analisi del valore limite potrebbe essere utilizzata per selezionare i dati di test quando un cliente è limitato nel numero di oggetti che può selezionare per l'acquisto.

In molte situazioni, i requisiti non funzionali possono essere documentati come user stories. Le tecniche di progettazione dei test black box (come analisi del valore limite) possono anche essere usate per creare test per caratteristiche qualitative non funzionali. La user story potrebbe contenere requisiti di prestazioni o di affidabilità. Ad esempio, una data esecuzione non può superare un limite di tempo o un dato numero di operazioni deve fallire meno di un certo numero di volte.

Per altre informazioni sull'uso delle tecniche black box di progettazione dei test, si veda il syllabus a livello Foundation [ISTQB_ALTA_SYL].

3.3.4 Test esplorativo e test Agile

Il Test esplorativo è importante nei progetti Agile a causa del tempo limitato a disposizione per l'analisi del test e i dettagli limitati delle user stories. Al fine di 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, quali test analitico basato sul rischio, test basato su requisiti analitici, test model-based e test di non regressione. Le strategie di test e l'uso di strategie di test miste sono discussi nel syllabus a livello Foundation [ISTQB_FL_SYL].

Nel test esplorativo, la progettazione e l'esecuzione dei test si effettuano allo stesso tempo, guidati da test charter. Una test charter fornisce le condizioni di test da coprire durante una sessione di test in un tempo determinato. Durante il test esplorativo, i risultati dei più recenti test guidano il test successivo. La stesse tecniche white box e black box possono essere utilizzate per progettare i test, come quando si eseguono test pre-progettati.

Una test charter può includere le seguenti informazioni:

- Attore: l'utente del sistema
- Obiettivo: il tema della charter, incluso quale particolare obiettivo l'attore vuole ottenere, cioè le condizioni di test.
- Setup: cosa deve essere pronto per iniziare l'esecuzione del test
- Priorità: importanza relativa del charter, in base alla priorità associata alla user story o al livello di rischio.
- Riferimenti: specifiche (per es. user story), rischi o altre fonti di informazione.

- Dati: quali dati sono necessari pe eseguire la charter
- Attività: una lista di idee su ciò che l'attore può voler fare con il sistema (per es.: "Loggarsi al sistema come super user") e su ciò di cui deve interessarsi il test (sia test positivi che negativi)
- Oracolo : come valutare il prodotto per determinare i risultati corretti (per es.: catturare cosa accade sullo schermo e confrontarlo a ciò che è scritto sul manuale utente)
- Variazioni: azioni alternative e valutazioni per completare le idee descritte nelle attività.

Per gestire i test esplorativi, può essere utilizzato un metodo chiamato gestione dei test basato sulla sessione. Una sessione è definita come un periodo ininterrotto di test 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 profonda (casi limite, scenari, interazioni).

La qualità dei test dipende dalla capacità del tester di porre domande pertinenti su cosa testare. Alcuni esempi sono i seguenti:

- Che cosa è più importante da scoprire sul sistema?
- In che modo il sistema potrebbe fallire?
- Cosa succede se .....?
- Che cosa dovrebbe accadere quando .....?
- Sono soddisfatte le esigenze dei clienti, i requisiti e le aspettative?
- E' possibile installare il sistema (e rimuoverlo se necessario) in tutti i percorsi di aggiornamento?

Durante l'esecuzione dei test, il tester utilizza la creatività, l'intuizione, le conoscenze e le competenze per trovare i possibili problemi del 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 test. Un euristica può guidare il test su come eseguire il test e per valutare i risultati. I relativi esempi comprendono:

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

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

- Copertura del test: quali dati di input sono stati utilizzati, quanto è stato coperto e quanto resta da testare
- Note di valutazione: osservazioni durante il test, se il sistema e la funzionalità in test sembrano essere stabili, dove sono stati trovati eventuali difetti, ciò che è previsto come prossimo passo secondo le attuali osservazioni ed ogni altra lista di idee
- Elenco dei rischi/strategia: quali rischi sono stati coperti e quali rimangono tra i più importanti, sarà seguita la strategia iniziale, ha bisogno di eventuali modifiche

- Problemi, domande e anomalie: qualsiasi comportamento inaspettato, tutte le domande riguardanti l'efficienza del metodo, tutte le preoccupazioni circa i tentativi di test, l'ambiente di test, i dati di test, le incomprensioni sulla funzione, gli script di test o il sistema sotto test.
- il comportamento reale: la registrazione del comportamento effettivo del sistema che deve essere salvato (ad esempio, video, schermate, file di dati di output)

Le informazioni registrate dovrebbero essere conservate e/o riassunte utilizzando strumenti di gestione dello stato (ad esempio, strumenti di gestione di test, strumenti di gestione delle attività, la task board), in un modo che renda più facile per gli stakeholders capire lo stato attuale per tutti i test eseguiti.

3.4 Strumenti nei progetti Agile

Gli strumenti descritti nel Syllabus Foundation Level [ISTQB_FL_SYL] sono usabili anche in un progetto Agile. Non tutti gli strumenti sono utilizzati allo stesso modo e alcuni strumenti hanno più rilevanza per i progetti Agile di quello che hanno nei progetti tradizionali. Ad esempio, anche se gli strumenti di gestione dei test, gli strumenti di gestione dei requisiti e gli strumenti di gestione degli incidenti (strumenti di defect tracking) possono essere utilizzati dai Team Agile, alcuni Team Agile optano per uno strumento all-inclusive (ad esempio, per la gestione del ciclo di vita dell'applicazione o la gestione delle attività) che fornisca funzioni utili per lo sviluppo Agile, come task boards, grafici burndown e user stories. Gli strumenti di gestione della configurazione sono importanti per i tester in un Team Agile a causa del numero elevato di test automatizzati a tutti i livelli e la necessità di memorizzare e gestire i prodotti dei test automatizzati associati.

Oltre agli strumenti descritti nel syllabus Foundation Level [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 all'interno del Team e la condivisione delle informazioni, che sono fondamentali per le pratiche Agile.

3.4.1 Strumenti per la gestione e la tracciamento delle attività

In alcuni casi, i Team Agile utilizzano story/task board fisici (ad esempio, lavagne, pannelli) per gestire e tracciare le user stories, i test e le altre attività nel corso di ogni sprint. Altri Team usano del software per la gestione del ciclo di vita delle applicazioni e per la gestione delle attività, comprese le task board elettroniche. Questi strumenti servono per i seguenti scopi:

- Memorizzare le user stories e i relativi compiti di sviluppo e test, per garantire che nulla venga perso durante uno sprint
- Registrare le stime dei membri del Team sulle loro attività e calcolare automaticamente lo sforzo necessario per implementare una storia, per sostenere sessioni di pianificazione dell'iterazione efficienti
- Associare attività di sviluppo e di test con la stessa user story, per fornire un quadro completo dello sforzo del Team necessario per implementare la user story
- Aggregare gli aggiornamenti degli sviluppatori e dei tester allo stato delle attività quando completano il loro lavoro, fornendo automaticamente un'istantanea calcolata dello stato di ogni user story, dell'iterazione e della release globale
- Fornire una rappresentazione visiva (tramite metriche, grafici e cruscotti), dello stato attuale di ogni user story, dell'iterazione e della release, consentendo a tutte le parti interessate, comprese le persone in team distribuiti geograficamente, di controllare rapidamente lo stato
- Integrarsi con gli strumenti di gestione della configurazione, il che può consentire la registrazione automatica di check-in del codice e delle build e, in alcuni casi, aggiornare automaticamente lo stato delle attività.

3.4.2 Strumenti per la comunicazione e la condivisione delle informazioni

Oltre a e-mail, documenti e comunicazioni verbali, i Team Agile utilizzano spesso tre ulteriori tipi di strumenti per supportare la comunicazione e la condivisione delle informazioni: wiki, instant messaging e la condivisione del desktop.

I wiki permettono ai Team di costruire e condividere una base di conoscenza on-line su vari aspetti del progetto, inclusi i seguenti:

- Diagrammi delle funzionalità del prodotto, discussioni sulle funzionalità, diagrammi prototipali, foto della lavagna e altre informazioni
- Strumenti e/o altre tecniche per sviluppare e testare che sono utili agli altri membri del Team
- Metriche, grafici, lavagne sullo stato del prodotto, che è utile specialmente quando il wiki è integrato con gli altri strumenti, come il server delle build e il sistema di gestione delle attività, poiché lo strumento può aggiornare lo stato del prodotto automaticamente.
- Le conversazioni tra i membri del Team, come la messaggistica istantanea e le e-mail, ma in un modo che sia condiviso con tutti gli altri membri del Team.

Strumenti di messaggistica istantanea, di teleconferenza audio e chat video portano i seguenti benefici:

- Consentono in tempo reale la comunicazione diretta tra i membri del Team, specialmente per i team distribuiti
- Coinvolgono i team distribuiti nelle riunioni
- Riducono le bollette telefoniche con l'uso della tecnologia di voice-over-IP, rimuovendo i vincoli di costo che potrebbero ridurre la comunicazione tra i membri di team distribuiti

Gli strumenti di condivisione e di acquisizione di immagini del desktop forniscono i seguenti vantaggi:

- Nei team distribuiti, possono essere utilizzati per dimostrazioni sul prodotto, per revisioni del codice e anche il lavoro in coppia
- Acquisire dimostrazioni del prodotto alla fine di ogni iterazione, che possono essere inviate al wiki del Team

Questi strumenti devono essere usati per completare e estendere, non rimpiazzare, la comunicazione faccia a faccia 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 quotidiana è una pratica fondamentale in un Team Agile. Ciò richiede l'utilizzo di strumenti di integrazione continui e di strumenti di distribuzione delle build. Gli usi, i benefici e i rischi di questi strumenti sono stati descritti nel Paragrafo 1.2.4.

3.4.4 Strumenti di gestione della configurazione

In un Team Agile, gli strumenti di gestione della configurazione possono essere utilizzati non solo per memorizzare il codice sorgente e i test automatizzati, ma sono spesso memorizzati nello stesso repository del codice sorgente del prodotto anche i test manuali e gli altri prodotti di lavoro del test . Ciò consente la tracciabilità di quali versioni del software sono state testate, e con quali particolari versioni dei test, e consente modifiche rapide senza perdere informazioni storiche. I principali tipi di sistemi di controllo della versione includono sistemi di controllo del codice sorgente centralizzati e sistemi di controllo della versione distribuiti. Le dimensioni del Team, la struttura, la posizione e requisiti di l'integrazione con altri strumenti determineranno quale sistema di controllo della versione è adatto per un particolare progetto Agile.

3.4.5 Strumenti per la progettazione, la realizzazione e l'esecuzione

Alcuni strumenti sono utili ai tester Agile in punti specifici nel processo di testing del software. Mentre la maggior parte di questi strumenti non sono nuovi o specifici per Agile, forniscono comunque funzionalità importanti dato il rapido cambiamento dei progetti Agile.

- Strumenti per la progettazione del test: l'uso di strumenti come le mappe mentali sono divenuti più popolari per accelerare la progettazione e la definizione dei test per una nuova funzionalità
- Strumenti di gestione dei casi di test: Il tipo di strumenti di gestione dei casi di test utilizzati nell'Agile possono far parte degli strumenti per la gestione del ciclo di vita dell'applicazione o per la gestione delle attività di 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 molti dati e combinazioni di dati sono necessari per testare l'applicazione. Questi strumenti possono anche aiutare a ridefinire la struttura del database quando un prodotto subisce cambiamenti durante un progetto Agile e adattare gli script per generare i dati. Questo consente un rapido aggiornamento dei dati di test, quando si verificano cambiamenti. Alcuni strumenti di preparazione dei dati di test utilizzano come origine i dati di produzione e utilizzano script per rimuovere o rendere anonimi i dati sensibili. Altri strumenti di preparazione dei dati di test possono aiutare a validare dati di input e output di grandi dimensioni
- Strumenti di caricamento dei dati di test: dopo che i dati sono stati generati per il test, è necessario che siano caricati nell'applicazione. L'immissione manuale dei dati spesso richiede tempo ed è soggetta ad errori, ma gli strumenti di caricamento dei dati sono a disposizione per rendere il processo affidabile ed efficiente. In realtà, molti degli strumenti per la generazione di dati includono una componente di caricamento dei dati integrata. In altri casi, è anche possibile il caricamento di massa utilizzando i sistemi di gestione di database.
- Strumenti di esecuzione dei test automatizzati: ci sono strumenti di esecuzione di test che sono più adatti al test Agile. Strumenti specifici sono disponibili sia del tipo commerciale che open source per supportare i vari approcci al test, come il Behaviour Driven Development, il Test Driven Development e l'Acceptance Test Driven Development. Questi strumenti permettono ai tester e al personale di staff di esprimere il comportamento del sistema previsto in tabelle o in linguaggio naturale utilizzando parole chiave.
- Strumenti di test esplorativi: strumenti che registrano le attività eseguite su una applicazione durante una sessione di test esplorativi sono utili per i tester e gli sviluppatori, poiché registrano le azioni eseguite. Questo è utile quando un difetto è trovato, poiché le azioni eseguite prima che si sia verificato l'errore sono state catturate e possono essere utilizzate per segnalare il difetto agli sviluppatori. La registrazione dei passi eseguiti in una sessione di test esplorativo può rivelarsi utile se il test è in seguito incluso nella suite di automazione di test di regressione.

3.4.6 Strumenti di Cloud Computing e virtualizzazione

La virtualizzazione permette ad una singola risorsa fisica (server) di operare come molte risorse più piccole separate. Quando si usano macchine virtuali o istanze cloud, i Team hanno un maggior numero di server disponibili per lo sviluppo e il test. 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 è più efficiente con le funzionalità di snapshot integrate nella maggior parte degli strumenti di virtualizzazione. Alcuni strumenti di gestione dei test ora utilizzano le tecnologie di virtualizzazione per effettuare snapshot del server nel momento in cui viene rilevato un errore, consentendo ai tester di condividere lo snapshot con gli sviluppatori per indagare il guasto.

4. Riferimenti

4.1 Standards

Questa sezione elenca le norme menzionate nel presente Syllabus.

- [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
[ISTQB_GLOSSARY]	ISTQB Glossary of Terms used in Software Testing, Version 2014

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

I termini trovati nel ISTQB Glossary sono identificati all’inizio di ogni capitolo. Per i termini comuni della metodologia Agile, ci siamo basati sulle seguenti risorse di Internet che forniscono le relative definizioni.

http://guide.Agilealliance.org/
http://whatis.techtargetcom/glossary
http://www.scrumalliance.org/

Incoraggiamo i lettori a controllare 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 siti Internet che contengono informazioni relative al test. 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.