Certified Professional for Requirements Engineering

RE@Agile Primer

Syllabus e guida allo studio

Lars Baumann, Peter Hruschka,
Kim Lauenroth, Markus Meuten,
Sacha Reis, Gareth Rogers,
François Salazar,
Hans-Jörg Steffe, Thorsten Weyer

v1.4.0 | 1 Luglio 2024, 2024

Termini di utilizzo

1. I soggetti individuali e i training provider possono utilizzare questo syllabus e la guida allo studio come base per i corsi, a condizione che il copyright sia riconosciuto e incluso nei materiali del corso. Chiunque utilizzi questo Syllabus e la guida allo studio a scopo pubblicitario deve chiedere un consenso scritto a IREB.
2. Ogni persona o gruppo di persone può utilizzare questo syllabus e la guida allo studio come base per articoli, libri o altre pubblicazioni a patto che venga riconosciuto in esse il copyright degli autori e di IREB e V. come fonte e il proprietario di questo documento sia riconosciuto in tali pubblicazioni.

© IREB e.V.

Tutti i diritti riservati. Nessuna parte di questa pubblicazione può essere riprodotta, memorizzata in un sistema accessibile o trasmessa in qualsiasi forma o con qualsiasi mezzo, elettronico, meccanico, fotocopia, registrazione o altro, senza la preventiva autorizzazione scritta degli autori o IREB e.V.

Crediti e ringraziamenti

La versione originale in inglese del syllabus e della guida allo studio è stata scritta da: Lars Baumann, Peter Hruschka, Kim Lauenroth, Markus Meuten, Sacha Reis, Gareth Rogers, François Salazar, Thorsten Weyer.

Commenti per la revisione sono stati forniti da : Bernd Aschauer, Thomas Emmerich, Dirk Fritsch, Rainer Grau, Andrea Hermann, Krystian Kaczor, Niko Kaintantzis, Elisabeth Larson, Ladislau Szilagyi, Daniel Tobler, Erik van Veenendaal, Arun Vetrivel, Sven van der Zee.

La traduzione in Italiano è stata effettuata da Salvatore Reale e da Marco Sogliani. La revisione Italiana è stata effettuata da Luisa Mich.

Si ringraziano tutti coloro che hanno collaborato.

Versione originale in inglese approvato per rilascio il 2 Marzo 2017 dall'IREB Council su raccomandazione di Xavier Franch.

Il copyright © 2016–2024 per questo syllabus e guida allo studio è attribuito agli autori elencati precedentemente. I diritti sono stati trasferiti a IREB International Requirements Engineering Board e.V.

Scopo del documento

Questo syllabus e guida allo studio descrive il livello foundation della certificazione "RE@Agile" definito dall'International Requirements Engineering Board (IREB). Il syllabus con la guida allo studio forniscono ai training providers le basi per la creazione del proprio materiale per la formazione. Gli studenti possono utilizzare il syllabus con la guida allo studio per prepararsi all'esame.

Contenuto del syllabus e guida allo studio

Il livello foundation risponde alle esigenze di tutte le persone coinvolte nelle tematichedi Requirements Engineering e di Agile. Si tratta di persone che ricoprono ruoli quali la gestione di progetti o IT, esperti di dominio, analisti di sistema e sviluppatori di software, Scrum Master, Product Owner e persone che fanno parte di organizzazioni Agile.

Obiettivo del Syllabus

RE@Agile è allineato sia con la visione IREB di valori Agile, sia con la visione Agile dei valori di Requirements Engineering (Ingegneria dei Requisiti). Il suo contenuto include la classificazione e la valutazione dei prodotti del lavoro e delle tecniche di Requirements Engineering nel contesto Agile, dei prodotti del lavoro e tecniche Agile nel contesto di Requirements Engineering e di elementi essenziali del processo di sviluppo Agile. RE @ Agile risponde alle motivazioni per utilizzare Agile in un processo di sviluppo.

Un tema molto importante è la sinergia tra Requirements Engineering e Agile: i principi Agile relativi al Requirements Engineering e all'approccio Agile in relazione ai valori di base di Requirements Engineering.

Le certificazioni RE@Agile dell'IREB vogliono supportare i seguenti gruppi di persone:

- Requirements Engineer (ingegneri dei requisiti) che desiderano essere coinvolti nello sviluppo Agile e che mirano ad applicare con successo le loro tecniche in questo ambiente.
- Requirements Engineer che vogliono applicare concetti e tecniche consolidati derivanti da approcci Agile per migliorare i propri processidi Requirements Engineering.
- Professionisti Agile che desiderano apprendere il valore e i vantaggi della disciplina Requirements Engineering nei progetti Agile.
- Professionisti Agile che desiderano migliorare lo sviluppo Agile utilizzando tecniche e metodi collaudati di Requirements Engineering.
- Persone provenienti da discipline affini – responsabili IT, tester, sviluppatori, architetti e altri rappresentanti del mondo aziendale coinvolti nello sviluppo (principalmente, ma non solo, sviluppo del software) – che desiderano comprendere come integrare con successo gli approccidi Requirements Engineering e approcci Agile nei processi di sviluppo.

Livello di dettaglio

Il livello di dettaglio di questo syllabus e guida allo studio garantisce la consistenza dei corsi e degli esami a livello internazionale. Per raggiungere tale obiettivo, il syllabus contiene i seguenti:

- Obiettivi formativi generali
- Contenuti con una descrizione degli obiettivi formativi
- Riferimenti a ulteriore documentazione (se necessari)

Obiettivi educativi / livelli di conoscenza

Ad ogni modulo del syllabus è assegnato un livello di conoscenza. Un livello elevato include sotto-livelli. Le formulazioni degli obiettivi formativi sono espresse usando i verbi "conoscere" per il livello L1 e "dimostrare padronanza" per il livello L2. Questi due verbi riassumono i seguenti:

A ogni modulo e obiettivo formativo di questo Syllabus viene assegnato un livello cognitivo. Vengono utilizzati i seguenti livelli cognitivi:

- **L1: Conoscere** (descrivere, enumerare, caratterizzare, riconoscere, nominare, ricordare, ...) – ricordare o recuperare materiale appreso in precedenza.
- **L2: Capire** (spiegare, interpretare, completare, riassumere, giustificare, classificare, confrontare, ...) – cogliere/costruire il significato da materiale o situazioni specifiche.
- **L3: Applicare** (specificare, scrivere, progettare, sviluppare, implementare, ...) – applicare la conoscenza e le competenze in determinate situazioni.

Si noti che un obiettivo di apprendimento al livello di conoscenza cognitiva Ln contiene anche elementi di tutti i livelli cognitivi sottostanti (da L1 a Ln-1).

Esempio:
Un obiettivo di apprendimento del tipo "Applicare la tecnica RE xyz" si colloca al livello di conoscenza cognitiva (L3). Tuttavia, la capacità di applicare richiede che gli studenti conoscano la tecnica RE xyz (L1) e che capiscano a cosa serve la tecnica (L2).

Tutti i termini definiti nel glossario devono essere conosciuti (L1), anche se non sono esplicitamente menzionati negli obiettivi formativi. Il glossario è disponibile per il download nella homepage IREB all'indirizzo https://www.ireb.org/en/downloads/#re-agile-glossary

Questo syllabus e guida allo studio usa l'abbreviazione "RE" per Requirements Engineering.

Struttura del syllabus e guida allo studio

Il syllabus e guida allo Studio consiste di 4 capitoli principali. Ogni capitolo copre una unità educativa (EU). I titoli di ogni capitolo principale contengono il livello cognitivo del capitolo stesso, che corrisponde al livello più alto dei relativi sottocapitoli. Viene inoltre suggerito il tempo di insegnamento minimo che un corso dovrebbe pianificare per tale capitolo. I termini importanti del capitolo, che sono definiti nel glossario, sono elencati all'inizio del capitolo.

Esempio:	Capitolo 2 Fondamenti di RE@Agile (L1)
Durata:	1 ¼ ora
Termini:	Product Owner, Product Backlog, Sprint Backlog, Epic, User Story, Story Map

Questo esempio mostra che il capitolo 2 contiene obiettivi educativi a livello L1 e che sarebbero necessari 75 minuti per insegnare il materiale del capitolo.

Ogni capitolo può contenere sotto-capitoli. Il loro titolo include anche il livello cognitivo del loro contenuto.

Gli obiettivi formativi (EO, Educational Objectives) sono elencati prima del testo. La numerazione mostra a quale sottocapitolo appartengono.

Esempio: EO 3.1.2
Questo esempio mostra che l'obiettivo formativo EO 3.1.2 è descritto nel sotto-capitolo 3.1

L'Esame

Questo syllabus cone guida allo studio è la base per gli esami RE@Agile Primer. Sono disponibili due differenti esami:

- Esame con osservatore con domande a scelta multipla per ottenere la certificazione RE @ Agile Primer ufficiale. L'esame è simile agli esami con domande a scelta multipla CPRE Foundation Level e CPRE Modul RE@Agile Practitioner, ma con durata ridotta a 40 minuti.
- Auto-valutazione online con domande a scelta multipla e rilascio di conferma di partecipazione.

Gli esami con osservatore possono essere sostenuti al termine di un corso di formazione, ma anche indipendentemente dalla frequenza di corsi (ad esempio in un centro d'esame). La lista dei centri d'esame riconosciuti (Examination Provider) può essere consultata nella pagina Internet https://www.ireb.org/exams/bodies.

L'auto-valutazione sarà disponibile nella homepage IREB: http://www.ireb.org

Una domanda d'esame può coprire il contenuto di più capitoli del syllabus e guida allo studio. Tutti i capitoli (da EU 1 a EU 4) del syllabus e guida allo studio possono essere oggetto d'esame.

Storia delle Versioni

Versione	Data	Commenti
1.0.0	17 Settembre 2018	Versione iniziale, basato sulla versione Inglese 1.0.2
1.2.0	Aprile 2023	▪ Soluzione di errori ▪ Allineamento alla Scrum guide 2020 ▪ Allineamento alla versione 3 del CPRE Foundation Level e alla versione 2 del CPRE Advanced Level RE@Agile ▪ Livelli cognitivi di conoscenza allineati alla nuova definizione applicata in IREB ▪ Obiettivi educativi allineati ai nuovi livelli cognitivi di conoscenza ▪ Tempo di insegnamento raccomandato delle unità educative modificato per essere allineato ai nuovi livelli cognitivi di conoscenza
1.3.0	Settembre 2023	Correzione di errori minori
1.4.0	1 Luglio 2024	Correzione dei riferimenti interrotti, implementazione del nuovo Corporate Design

Contenuto

La Visione di RE@Agile

Motivazioni e Presupposti

La qualità dei requisiti determina il successo o il fallimento dell'intero processo di sviluppo di un prodotto, indipendentemente dalla metodologia di sviluppo applicata. Contrariamente alla credenza diffusa, le tecniche e i metodi di Requirements Engineering funzionano a prescindere dal loro utilizzo all'interno di specifiche metodologie di sviluppo (come waterfall o Scrum). Tuttavia, Requirements Engineering è comunemente percepito come una disciplina di sviluppo non Agile, il che porta all'idea errata che il corpus di conoscenze di Requirements Engineering non sia rilevante per il successo dei processi di sviluppo Agile.

In molti casi, gli approcci di Requirements Engineering e Agile sono considerati separatamente piuttosto che insieme. Mentre nei processi di sviluppo convenzionali il Requirements Engineering è strutturato con ruoli dedicati come se si trattasse di una disciplina separata all'interno del ciclo di vita di un sistema, nello sviluppo Agile l'importanza di Requirements Engineering è spesso sottovalutata.

Gli approcci Agile si basano sulla comunicazione diretta, sulla semplicità delle soluzioni e sul feedback (riscontro). Uno dei loro valori principali è la risposta rapida alle modifiche. Pertanto, le modifiche ai requisiti e alle loro priorità rappresentano un concetto intrinseco di tutti gli approcci Agile. D'altra parte, dare la giusta importanza alle competenze in Requirements Engineering nei processi di sviluppo Agile permetterebbe di sfruttare il successo dei progetti Agile aumentando in modo sostenibile la qualità dei sistemi e dei prodotti sviluppati. Nel contempo Requirements Engineering potrebbe beneficiare in modo significativo di alcuni principi e tecniche Agile molto utili, indipendentemente dalla specifica metodologia di sviluppo applicata.

Attualmente, in molti casi, vi sono esperti o nel Requirements Engineering o nell'applicazione di alcuni approcci Agile. Di conseguenza, gli esperti di entrambe le parti devono trovare la migliore modalità per sfruttare i benefici derivanti dall'utilizzo di principi e tecniche dell'altra area di competenza. Certificazioni incentrate sull'integrazione di entrambi i campi di competenza non sono attualmente disponibili, né nella comunità di Requirements Engineering né nella comunità Agile. Per questo è molto positivo costruire un collegamento condiviso tra Requirements Engineering e approcci Agile e quindi anche tra requirements engineer ed esperti Agile, in modo che entrambi possano comunicare efficacemente ed efficientemente tra loro.

La riposta IREB a questa esigenza è la Certificazione RE@Agile.

Relativamente a RE@Agile

Un tale collegamento dovrebbe essere costruito da due direzioni diverse: da un lato la comunità di Requirements Engineering deve capire come applicare con successo le proprie tecniche e metodi nei processi di sviluppo Agile, nonché come applicare tecniche specifiche derivanti dagli approcci Agile allo scopo di migliorare la pratica di Requirements Engineering. D'altra parte, poiché gli approcci Agile mirano a fornire software di valore il più presto possibile, i professionisti Agile devono capire come sfruttare questa caratteristica, applicando concetti e tecniche consolidati derivanti dal Requirements Engineering.

I principi base di RE@Agile sono:

- *Gli approcci di Requirements Engineering e di Agile possono influenzarsi a vicenda*
  RE@Agile analizza i possibili vantaggi e le insidie delle tecniche di Requirements Engineering e di Agile. A tal fine, RE@Agile indirizza all'utilizzo di prodotti del lavoro e tecniche dalla disciplina di Requirements Engineering nei processi Agile, nonché all'utilizzo di prodotti del lavoro, ruoli e tecniche da approcci Agile nei processi di Requirements Engineering, nell'ambito di diverse metodologie di sviluppo.
- *Processi snelli e altamente flessibili*
  Basandosi sulla filosofia di RE@Agile, la differenziazione tra processi di sviluppo predittivi e adattativi è di vitale importanza. RE@Agile propone l'idea di un approccio snello e altamente adattativo per lo svolgimento di attività di Requirements Engineering nell'ambito dello sviluppo Agile. In RE@Agile, il Requirements Engineering è una disciplina di base piuttosto che una singola fase del processo: un processo continuo che deve essere eseguito sistematicamente e che richiede un alto livello di competenza ed esperienza.
- *Stretta collaborazione all'interno del team e con le principali parti interessate (stakeholders) e requisiti just-in-time*
  La comunicazione frequente e la stretta collaborazione tra tutti i membri del team e gli stakeholder principali sono di particolare importanza per il successo dei processi di sviluppo Agile. In RE@Agile, il team, insieme ai principali stakeholder, elabora, analizza, affina e documenta i requisiti in modo altamente interattivo. RE@Agile supporta chi pratica nella scelta delle attività giuste al momento giusto per garantire requisiti di alta qualità prima della loro implementazione.
- *Elicitazione dei requisiti situazionali e selettivi, analisi, specifica e affinamento*
  RE@Agile si basa sull'idea che non tutti i requisiti devono essere specificati con precisione e in modo molto dettagliato prima che inizi l'implementazione del sistema. Solo i requisiti che sono eccessivamente complessi (cioè non sono comprensibili per gli stakeholder o per gli sviluppatori) o critici (cioè non possono rischiare di essere fraintesi) sono affinati e specificati in modo più dettagliato. Il processo complessivo si basa sulla filosofia condivisa che le modifiche ai requisiti funzionali sono ben accette e facili da soddisfare.

- *Evitare attività e funzionalità meno rilevanti e garantire il Prodotto Minimo Utilizzabile*
  Uno dei principi Agile è la "Semplicità". Secondo questo principio, la prima fase dello sviluppo del sistema o del prodotto nei processi Agile è spesso l'MVP (Minimum Viable Product, o Prodotto Minimo Utilizzabile). L'MVP è un sistema definito e rilasciabile, che offre solo un insieme di funzionalità di base, ma che fornisce agli utenti finali un valore aziendale sufficiente a consentire un'apprendimento validato. L'ambito ridotto dell'MVP consente l'eliminazione degli sprechi durante lo sviluppo e permette di avere un rapido feedback dai clienti. Una delle versioni successive del prodotto è spesso il MMP (Minimum Marketable Product, o Prodotto Minimo Commercializzabile) – un prodotto con il più piccolo insieme di funzionalità che risponde alle esigenze degli utenti e che quindi ha un valore di mercato. RE@Agile fornisce risposte a due domande molto importanti, che sono abbastanza rilevanti anche in processi di sviluppo non-Agile: "Come semplificare la gestione dei rilasci e il processo di definizione del prodotto?" e "Come definire l'MVP o l'MMP in base ai requisiti?"

Relativamente alla Certificazione IREB RE@Agile:

La conoscenza del CPRE Foundation Level e dei processi di sviluppo Agile è consigliata come prerequisito.

Il certificato RE@Agile Primer è destinato a professionisti di discipline affini: Project manager, business analyst, architetti, sviluppatori, tester e anche altri ruoli legati al business. Questo certificato si focalizza sulla comunicazione tra i Requirements Engineer e gli esperti di Agile, nonché sulla comprensione dei termini di entrambe le aree. I titolari dei certificati possono dialogare con esperti Agile sul Requirements Engineering e con gli specialisti in Requirements Engineering sugli approcci Agile e sullo sviluppo Agile.

Una persona con il Certificato RE@Agile Primer:

- ha familiarità con la terminologia appropriata di Requirements Engineering e con gli approcci Agile
- comprende il ruolo e l'importanza di Requirements Engineering nei processi di Agilità, nonché il valore dei principi Agile nel Requirements Engineering

1 Motivazione e visione (L1)

Durata: 1 ½ ora

Termini: Valori, Agile Manifesto, Pratiche, Attività, Sprint, Agile

Obiettivi Formativi

EO 1.1.1 Conoscenza delle motivazioni per utilizzare metodi Agile
EO 1.2.1 Conoscere gli obiettivi dell'ingegneria dei requisiti secondo l'IREB
EO 1.2.2 Conoscenza dei Valori Base del Manifesto Agile e dei Principi da esso derivanti
EO 1.3.1 Conoscenza delle differenze fra principi, pratiche, e attività
EO 1.3.2 Conoscere le differenze tra la mentalità Agile e la mentalità dell'Ingegneria dei requisiti
EO 1.3.3 Conoscenza delle sinergie fra approcci mentali e valori riguardanti RE@Agile
EO 1.3.4 Conoscenza del significato di "Documentazione" in un contesto Agile (in accordo con il Manifesto Agile)
EO 1.4.1 Conoscenza dei benefici, insidie e pregiudizi nell'utilizzo di RE@Agile
EO 1.4.2 Conoscenza di esempi di pregiudizi
EO 1.5.1 Sapere che i valori Agile possono essere trasferiti nel lavoro concettuale
EO 1.5.2 Conoscenza di contesti esemplificativi che favoriscono l'applicazione dell'approccio Agile nel lavoro concettuale

1.1 Motivazioni all'utilizzo di Agile (L1)

Diversi studi (vedi [MeMi2015]) mostrano che l'industria dell'information technology nel suo insieme sta subendo un cambiamento essenziale: l'information technology sta diventando un driver importante in diverse aree di business (ad esempio commercio elettronico, social media) e in settori tecnici (ad esempio l'industria automobilistica o avionica). Di conseguenza, i sistemi e i prodotti nei settori di business guidati dall'IT devono subire un costante adeguamento per tenere il passo con le mutevoli esigenze dei clienti o del mercato. Non appena si verifica un cambiamento nel mercato, i sistemi devono essere adattati in base a tali modifiche.

I metodi di sviluppo esistenti incentrati sulla prevedibilità e sulla stabilità a lungo termine non sono stati ideati per tali circostanze e spesso falliscono in situazioni di mercati o progetti in rapida evoluzione. I metodi Agile, guidati dal manifesto Agile (vedi 1.2), si sono diffusi per colmare questa lacuna. Agile (o l'Agilità) è di per sé un termine difficile e può essere definito come segue (vedi [ShYo2006]):

> Un rapido movimento di tutto il corpo con cambio di velocità o direzione in risposta a uno stimolo.

Questa definizione non deriva dall'ingegneria del software, ma dallo sport e riflette la motivazione essenziale per l'utilizzo di un metodo Agile: se la situazione del mercato o del progetto richiede cambiamenti rapidi e controllati, sono adatti i metodi Agile. Un metodo Agile è, ovviamente, più che un semplice sviluppo rapido (vedi 1.2), ma in sostanza tutti i principi alla fine si concentrano su rilasci frequenti a un dato livello di qualità.

Ciò si traduce in cicli di feedback frequenti, che a loro volta consentono di rispondere rapidamente alle esigenze dei clienti.

È importante riconoscere che né i metodi né l'approccio Agile sono fini a se stessi [Meyer2014]. Un'organizzazione deve essere in grado di selezionare l'approccio di sviluppo adeguato che si adatti alle esigenze del proprio mercato, dei clienti e dell'organizzazione. Gartner afferma addirittura che la capacità di sviluppare l'IT con l'approccio giusto è il principale fattore di successo per il business digitale [MeMi2015].

1.2 Visione e Valori in RE e Agile (L1)

La mentalità e i valori della RE sono enunciati nella definizione IREB di Requirements Engineering (vedi [Glinz2022]): L'approccio sistematico e disciplinato alla specifica e alla gestione dei requisiti con l'obiettivo di comprendere i desideri e le esigenze delle parti interessate e di ridurre al minimo il rischio di fornire un sistema che non soddisfi tali desideri ed esigenze.

In RE, parliamo di un sistema anziché di un software o di un prodotto. L'uso del termine sistema non intende escludere prodotti, altri tipi di software o anche altre entità (ad esempio processi aziendali o hardware). RE preferisce il termine sistema perché esso enfatizza il fatto che un sistema è un insieme di parti o elementi che interagiscono in un ambiente. RE chiama l'ambiente il contesto del sistema. Nel syllabus il termine sistema verrà usato intendendo che esso include prodotti e ogni altro tipo di elementi correlati al software.

L'IREB FL [CPREFL2022] definisce inoltre un insieme di quattro attività principali di RE: elicitazione, documentazione, validazione/negoziazione e gestione dei requisiti. Questo elenco di attività non denota un insieme specifico di passi o una sequenza in cui esse vengono eseguite. Un valore fondamentale di IREB FL è che RE è un approccio di processo agnostico: RE fornisce un ricco corpo di conoscenze costituito da vari metodi e da una ricca collezione di tecniche che possono essere applicate in qualsiasi approccio di sviluppo. Esso non consiglia o specifica alcun processo.

Questo syllabus e guida allo studio userà il termine "metodi Agile" per riferirsi al ricco insieme di approcci che sono emersi nel campo di Agile (vedi 2). Per distinguere i metodi Agile da altri metodi di sviluppo (ad esempio plan-driven o waterfall"), nel syllabus si utilizza il termine "metodi non-Agile". Questi due termini prescindono da una valutazione di quale sia il migliore - IREB è convinto che entrambi i tipi di metodi (Agile e non-Agile) abbiano il loro valore.

La visione di Agile è definita dal manifesto Agile e dai dodici principi che la sostengono (vedi [AgileMan2001]):

Manifesto Agile

Stiamo scoprendo modi migliori di creare software,
sviluppandolo e aiutando gli altri a fare lo stesso.

Grazie a questa attività siamo arrivati a considerare importanti:

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,
consideriamo più importanti le voci a sinistra.

Principi Agile

1. La nostra massima priorità è soddisfare il cliente rilasciando software di valore, fin da subito e in maniera continua.
2. Accogliamo i cambiamenti nei requisiti, anche a stadi avanzati dello sviluppo. I processi agili sfruttano il cambiamento a favore del vantaggio competitivo del cliente.
3. Consegnamo frequentemente software funzionante, con cadenza variabile da un paio di settimane a un paio di mesi, preferendo i periodi brevi.
4. Committenti e sviluppatori devono lavorare insieme quotidianamente per tutta la durata del progetto.
5. Fondiamo i progetti su individui motivati. Diamo loro l'ambiente e il supporto di cui hanno bisogno e confidiamo nella loro capacità di portare a termine il lavoro.
6. Una conversazione faccia a faccia è il modo più efficiente e più efficace per comunicare con il team ed all'interno del team.
7. Il software funzionante è il principale metro di misura di avanzamento.
8. I processi Agile promuovono uno sviluppo sostenibile. Gli sponsor, gli sviluppatori e gli utenti dovrebbero essere in grado di mantenere indefinitamente un ritmo costante.
9. La continua attenzione all'eccellenza tecnica e alla buona progettazione esaltano l'Agilità.
10. La semplicità – l'arte di massimizzare la quantità di lavoro non svolto – è essenziale.
11. Le architetture, i requisiti e la progettazione migliori emergono da team che si auto-organizzano.
12. A intervalli regolari il team riflette su come diventare più efficace, dopo di ché regola e adatta il proprio comportamento di conseguenza.

Se confrontiamo i valori e la visioni di RE e Agile, non possiamo trovare alcuna parte che possa contraddire i valori o la visione dell'altro. Il valore più importante è condiviso da RE e Agile e cioè rendere soddisfatto del prodotto l'utente finale perché la soluzione soddisfa le sue necessità o risolve i suoi problemi principali.

Tuttavia, dobbiamo riconoscere che la visione e i valori di RE e Agile sono in qualche modo disgiunti. L'intersezione di RE e Agile è definita dal contenuto di RE@Agile, che sarà ulteriormente spiegata nella prossima UE.

1.3 Collegare RE e i principi Agile verso RE@Agile (L1)

Prima di entrare nei dettagli di RE@Agile, dobbiamo chiarire alcuni termini. Nell'ambito dell'industria del software e della ricerca, esiste una vasta conoscenza su come lavorare e comportarsi nello sviluppo del software. Questa conoscenza esiste a vari livelli di astrazione. Nel seguito, introdurremo la differenziazione fra principi, pratiche e attività come tre livelli di astrazione per parlare dello sviluppo di software (vedere [Meyer2014])

- Un principio è un'affermazione prescrittiva che è astratta e contestabile
- Una pratica o tecnica è un'istanza di un principio per un determinato contesto
- Un'attività è un'esecuzione reale o pianificata di una pratica

I termini chiave per differenziare le tre definizioni sono: prescrittivo, astratto e contestabile Prescrittivo significa che la dichiarazione indirizza l'azione anziché indicare un fatto o una proprietà. L'astrazione distingue un principio da una pratica. Ad esempio, "testare una funzionalità software prima della consegna" è prescrittiva e astratta, mentre "creare un test unitario per ogni funzionalità software" è una pratica basata sul principio dato. Un'attività per questo esempio sarebbe la creazione di uno unit test per la funzione di ricerca di un sistema di gestione biblioteche. Contestabilità significa che una persona con sufficiente conoscenza di base può essere in disaccordo con un principio. Il suddetto principio ("testa un ...") soddisfa questo criterio. Si potrebbe sostenere che il testing potrebbe non essere sufficiente per le funzionalità safety-critical, e che queste invece dovrebbero essere verificate con metodi matematici. Una dichiarazione che non è contestabile ("cercare di ottenere un'alta qualità") non dovrebbe essere considerata un principio per guidare il comportamento

Sapere i principi che stanno alla base delle nostre pratiche (che di per sé è un principio) porta a decisioni consapevoli sulle nostre azioni. Il rispetto delle diverse pratiche per soddisfare i principi ci dà la capacità di reagire in modo diverso a seconda della situazione data. La contestabilità promuove discussioni sull'applicabilità di un principio o di una pratica in un determinato contesto e quindi aiuta a decidere se applicare o meno un principio.

La conoscenza dei principi e delle pratiche è solo un primo passo: la corretta applicazione di una pratica è una competenza a sé stante. Per esempio, la definizione di use case (casi d'uso) è una pratica ampiamente conosciuta per il principio "specifica i requisiti funzionali per le feature importanti". Tuttavia, scrivere uno use case di alta qualità è una competenza a sé stante e conoscere gli elementi di un template di use case non è sufficiente.

In sostanza, ora abbiamo definito i tre livelli di competenze:

1. Conoscere i principi (corrisponde al livello cognitivo L1)
2. Comprendere le pratiche per realizzare i principi dati (corrisponde al livello cognitivo L2)
3. Applicare una pratica in un determinato contesto (capacità di eseguire un'attività con alta qualità) (corrisponde al livello cognitivo L3)

Confrontando la disciplina Agile con la disciplina RE (vedi 1.2), è possibile capire perché le due discipline siano a volte percepite essere in conflitto: RE si occupa dell'elicitazione sistematica e della documentazione dei requisiti come prodotti del lavoro a sé stanti, mentre Agile sottolinea l'importanza del software funzionante rispetto ad una documentazione completa e considera gli individui e la comunicazione più importanti dei processi e degli strumenti.

Una adozione esagerata di entrambe le visioni può, in pratica, portare a un conflitto: una assunzione errata di RE è che sia possibile creare un documento di requisiti completo, consistente e concordato, che possa essere implementato senza ulteriori modifiche. Una assunzione errata analoga di Agile è che un progetto di sviluppo possa iniziare senza alcun lavoro preparatorio e possa avere successo solo distribuendo software, a intervalli regolari, che venga revisionato dagli stakeholder [Nota: i clienti sono un sottoinsieme di stakeholder da un punto di vista RE] e migliorato in base al feedback.

Affermiamo che le visioni RE e Agile non sono affatto in conflitto: entrambi gli approcci condividono lo stesso obiettivo della consegna di software a un livello di qualità ben definito. I metodi Agile possono fornire software funzionante in modo efficiente e veloce (con tempo di ciclo ridotto). RE fornisce le tecniche adeguate per comprendere i desideri e le esigenze degli stakeholder per poter sviluppare il software giusto.

RE facilita:

- La comprensione dei desideri e delle necessità degli utenti per sviluppare software di valore (1^ principio Agile)
- L'utilizzo di strumenti utili per riconoscere i cambiamenti nel mercato per fornire un vantaggio competitivo agli stakeholder (2^ principio Agile)
- L'adozione di strumenti e tecniche appropriate per promuovere una collaborazione efficiente tra stakeholder e sviluppatori (4^ principio Agile)
- L'utilizzo di appropriati strumenti e tecniche per supportare la comunicazione verbale (6^ principio Agile)
- La comprensione opportuna dei desideri e delle necessità degli stakeholder per ridurre al minimo lo sviluppo di software non necessario (10^ principio Agile).

La differenza importante tra l'applicazione di RE in Agile e in altri metodi di sviluppo è la tempistica e il processo applicato. Con questo syllabus e guida allo studio, IREB definisce il contenuto di RE@Agile che mostra come applicare RE nel contesto dei metodi Agile.

IREB preferisce il termine RE@Agile al termine "Agile RE" per chiarire che RE è indipendente dal processo.

Il manifesto dello sviluppo software Agile sottolinea il valore di un incremento del software funzionante (o del prodotto funzionante) rispetto ad una documentazione completa. Un'interpretazione esasperata ha portato all'idea sbagliata che i metodi Agile abbiano abbandonato del tutto la documentazione. Questa interpretazione è errata: la documentazione che abbia uno scopo è ancora ben accetta e raccomandata in Agile, ma solo quella che supporta lo sviluppo o fa parte del prodotto.

Un problema percepito in passato è stato il fatto che molti progetti hanno creato documentazione senza uno scopo ben definito o un valore aggiunto: secondo i principi Agile questo tipo di documentazione deve essere evitata.

1.4 Vantaggi, pregiudizi e insidie dell'utilizzo di RE@Agile (L1)

RE@Agile può offrire molti vantaggi. Tuttavia, questi benefici non sono scontati: ci sono possibili pregiudizi e insidie che dovrebbero essere evitati.

1.4.1 Vantaggi di RE@Agile

Le competenze di RE & Dev nello stesso team possono ridurre i passaggi di consegne: la pratica di team inter-funzionali nei metodi Agile richiede che il team abbia tutte le competenze necessarie per lo sviluppo di un incremento di prodotto in base ai requisiti selezionati.

Eseguire le attività di RE all'interno del team può ridurre la necessità di creare una documentazione completa dei requisiti in anticipo, dato che i membri del team possono spiegare determinati dettagli direttamente ad altri membri. Il vantaggio dei documenti in questo contesto sarà legato alla documentazione dei risultati delle discussioni e la conservazione della conoscenza.

Lo sviluppo incrementale consente l'ottimizzazione delle idee pre-esistenti: il principio fondamentale dei metodi Agile è lo sviluppo incrementale del software in successive iterazioni. Il processo iterativo crea prodotti del lavoro (ad esempio un modello di processo aziendale, una epic, una user story, uno use case, un prototipo di interfaccia utente, una descrizione del processo o il software) e li migliora in una sequenza di attività di sviluppo e revisione. Il vantaggio di tale procedura è che la qualità dell'prodotto del lavoro o del software viene continuamente migliorata e ottimizzata. Inoltre, piccoli incrementi consentono discussioni anticipate con il cliente e riducono al minimo il rischio di grandi divari tra le aspettative dei clienti e il sistema sviluppato.

L'affinamento è un principio per maturare e validare i requisiti: nello sviluppo Agile è stata adottata un'ottima pratica – l'affinamento continuo. Consiste nel tenere riunioni periodiche di affinamento per consentire al team di sviluppo di revisionare e dettagliare i requisiti su base continua, il tutto in stretta comunicazione con gli stakeholder. Inoltre, la buona pratica della Definition of Ready viene utilizzata come elemento di qualità per verificare che un requisito sia pronto per l'implementazione in una successiva iterazione.

RE aiuta a definire un Product Backlog iniziale: RE fornisce diverse tecniche per acquisire una corretta comprensione dei requisiti degli stakeholder per il prodotto desiderato. RE fornisce quindi una comprensione più approfondita dei requisiti necessari per la definizione iniziale del Product Backlog. È importante sapere che tale attività di RE non crea una specifica dettagliata. Invece l'obiettivo di questa attività è incentrato su una comprensione completa del prodotto a un determinato livello di astrazione (ad es., comprensione e definizione degli use case essenziali o di epic e user story). Ad esempio, un complicato

processo di business di alto livello può essere scomposto in epic e user story usando metodi RE ben consolidati per creare un backlog iniziale [CPREALAGILE2022].

1.4.2 Pregiudizi di RE@Agile

Nel mondo dello sviluppo (principalmente dello sviluppo di software) esistono alcuni pregiudizi e insidie riguardo a RE che devono essere discussi:

Pregiudizio – RE è solo analisi anticipata: molto spesso, RE è considerato applicabile solo come un'attività anticipata. RE, in quanto disciplina, è in realtà indipendente dal processo e non impone di anticipare un'analisi completa. Invece le attività di RE possono essere svolte all'interno di un metodo Agile allo stesso modo con cui vengono eseguite altre attività (ad es. codifica o testing). RE è un'attività incorporata all'interno di ogni iterazione.

Pregiudizio – Anticipare è un male: la preparazione per lo sviluppo iterativo è una componente essenziale di qualsiasi progetto di sviluppo non banale. Il concetto di riflessione anticipata non implica di per sé un particolare ciclo di vita del software o una lunga fase di analisi o di documentazione: il come e il quando la riflessione anticipata verrà eseguita sarà determinato dal contesto del progetto. Chi pratica l'Agile non dovrebbe desumere dalla letteratura Agile che la riflessione anticipata sia a priori una cosa negativa; il manifesto e i principi sopra presentati non supportano tale interpretazione. Le pratiche Agile che riflettono il valore della riflessione anticipata sono la story mapping (vedi [Patt2014]), la prototipazione (vedi [Martin1991]) e il test-driven development (TDD) (vedi [Beck2003]).

Pregiudizio – RE uguale documentazione: RE è spesso associato solo ai documenti che produce. Tuttavia, i documenti sono un potenziale risultato di un'attività che crea conoscenza. Un buon RE include l'essere consapevoli del fatto che anche il miglior documento non è mai completamente esaustivo. Al contrario, un documento serve a supportare gli obiettivi definiti in 3.2: conformità legale, conservazione di informazioni preziose, facilitazione della comunicazione e supporto ai processi di approfondimento.

Pregiudizio: – le user story sono sufficienti: le user story sono un metodo popolare per catturare le esigenze degli stakeholder. Tuttavia esse hanno lo scopo di iniziare la comunicazione e non di rappresentare la specifica completa. Il percorso da un'esigenza non specificata verso un'esigenza completa è riassunto nella pratica "3C – Card, Conversation, Confirmation". Una visione più completa dei requisiti può essere raggiunta attraverso una combinazione di user story con altre tecniche RE consolidate, come context diagrams, prototipi, use cases e user journeys.

Pregiudizio – La documentazione è inutile, solo il codice ha un valore duraturo: sebbene sia senz'altro possibile che in alcuni progetti con processi particolarmente pesanti, una specifica eccessiva possa essere un problema, non è corretto concludere che tutta la documentazione sia inutile. La documentazione dei requisiti, proprio come la documentazione di progetto, di test o operativa, ha il suo posto in determinati contesti. Le documentazioni sono tutte ugualmente valide e necessarie come prodotti del processo di sviluppo di qualsiasi prodotto software sostenibile.

Pregiudizio: – Il software funzionante è l'unico modo per validare i requisiti: il manifesto Agile attribuisce più valore al "software funzionante rispetto ad una documentazione esaustiva". Quando si giunge alla validazione dei requisiti, una conclusione errata dalla precedente affermazione è che la validazione dei requisiti basata sul software funzionante sia sempre preferibile alla validazione di una forma documentata dei requisiti stessi. Il software come mezzo per validare i requisiti è preferibile se i costi e i rischi connessi a tale approccio di validazione sono accettabili rispetto al risultato. Se i costi o i rischi sono elevati, RE fornisce diversi strumenti che consentono un riscontro veloce e la validazione dei requisiti prima che venga scritta una singola riga di codice, ad es.: mock up dell'interfaccia utente o story board (vedere).

1.4.3 Insidie di RE@Agile

Insidia: – Trattare i requisiti come una tipologia uniforme di informazione: un errore tipico relativo a RE in tutti i contesti e metodi di implementazione è il considerare il 'requisito' come una tipologia uniforme di informazione. Sulla base di questo equivoco, la documentazione dei requisiti è generalmente considerata una perdita di tempo, poiché i requisiti cambiano così rapidamente da perdere validità non appena sono stati scritti. I requisiti NON sono una tipologia uniforme di informazioni; i requisiti possono essere specificati a vari livelli di dettaglio, di astrazione e di formato. Ad esempio, la visione del sistema o gli obiettivi di un sistema sono requisiti ad alto livello di astrazione con una durata generalmente lunga; un prototipo eseguibile è un mezzo per validare una serie di requisiti o per ottenere nuovi requisiti.

Insidia: – Perdere il quadro generale: i metodi Agile sono spesso fraintesi e implementati in modo da concentrarsi solo sugli argomenti che sono immediatamente visibili al team. Dal punto di vista dello sviluppatore, questo potrebbe essere considerato come un principio utile perché egli può concentrarsi sul lavoro a portata di mano e non essere distratto da argomenti a lungo termine. Tuttavia se tutti si concentrassero solo sul lavoro a portata di mano, il quadro generale e la prospettiva a lungo termine andrebbero persi. I metodi Agile sostenibili indirizzano le prospettive a lungo e a medio termine all'interno di sessioni dedicate (ad es. sessioni di affinamento, sessioni di road-mapping o workshop di visioning). Un rischio correlato consiste nel perseguire la soluzione senza approfondire il problema di business (spesso chiamato l'esigenza).

Insidia: – Sovraccaricare gli stakeholder con troppe informazioni: i prodotti del lavoro di RE possono avere un'alta densità di informazioni e essere creati molto velocemente in un team Agile. Questo approccio è spesso utilizzato in progetti innovativi o di alta tecnologia per i quali è difficile trovare esperti in materia. Tuttavia il dedicare l'iterazione completa per lavorare su prodotti del lavoro di RE pone un alto carico di revisione sugli stakeholder, richiedendo loro di comprendere specifiche troppo lunghe. Risultati migliori possono essere ottenuti con una combinazione adeguata di specifiche e sviluppo (prototipazione).

Insidia: – Elaborare ogni argomento in modo incrementale e iterativo: Non tutti gli argomenti relativi ai requisiti per un sistema dovrebbero essere sviluppati in un modo dettagliato e incrementale. Gli argomenti con complessità additiva (vedi [Meyer2014]) sono

adatti allo sviluppo incrementale. Esempi tipici di tali argomenti sono i processi che possono essere separati in elementi indipendenti (ad es., il processo di acquisto in un negozio online). Gli argomenti con una complessità intrinseca (vedi [Meyer2014]) non sono adatti allo sviluppo incrementale, perché ogni nuova vista sull'argomento porterà a una comprensione completamente nuova delle informazioni già note. Esempi sono rappresentati dai calcoli con parametri di input complessi e parametri di output semplici (come polizze assicurative o componenti di controllo di un motore).

Insidia: – Lo sviluppo incrementale può scoraggiare l'innovazione radicale o dirompente: il processo incrementale di Agile potrebbe non incoraggiare lo sviluppo di idee innovative e/o dirompenti dal momento che un determinato prodotto del lavoro (ad es. il software o una feature o funzionalità del software) viene tipicamente migliorato localmente (ad es. correggendo errori o aggiungendo elementi mancanti) una volta che è stato definito. Sebbene il Manifesto Agile apprezzi esplicitamente il cambiamento, i processi incrementali di Agile in genere supportano l'innovazione continua per prodotti e servizi. Le innovazioni radicali o dirompenti emergono attraverso il contributo di idee multiple e l'integrazione con idee esistenti. [LiOg2011]. Lo sviluppo di idee alternative in termini di software è generalmente considerato uno spreco in ambienti Agile (vedere il 10^ principio: – massimizzare il lavoro non svolto). Per ottenere innovazioni radicali o dirompenti, è necessario adottare pratiche aggiuntive, come idee di "lean startup" o approcci di "design thinking" presentati in 1.5.

L'insidia maggiore e più importante: – Agile e cambiamento culturale non vanno di pari passo: i valori Agile promuovono cambiamenti nel modo di lavorare delle organizzazioni e comportano una perdita di responsabilità su alcuni risultati a favore di una responsabilità collettiva. Agile promuove inoltre retrospettive continue sul comportamento del team per migliorare il suo modo di lavorare: il cambiamento continuo da parte del team e infine dell'intera organizzazione diventa inevitabile. Tali cambiamenti culturali (organizzativi) richiedono sia tempo che personale qualificato per guidare il team verso una nuova direzione.

1.5 RE@Agile e lavoro concettuale (L1)

Lo sviluppo Agile è arrivato in essere nel mondo dell'ingegneria del software per affrontare le sfide provenienti dal mondo all'esterno all'ingegneria del software (vedi 1.1). Tuttavia, queste sfide non sono state sperimentate solo dal mondo dell'ingegneria del software. Altri settori dell'industria e della società soffrivano di sfide simili, come clienti esigenti e cicli di innovazione più rapidi. Altri settori hanno sviluppato approcci per il lavoro concettuale (ad.es. creando concetti o specifiche di sistemi) che erano abbastanza simili allo sviluppo Agile. Molti di questi sono particolarmente utili in una prospettiva RE per lo sviluppo di innovazioni e visioni di prodotto. Tali approcci saranno brevemente trattati ora, includendo una discussione sulla loro corrispondenza con la visione Agile (vedi 1.2). Presenteremo tre approcci come esempi di Agilità nel lavoro concettuale.

Design Thinking (vedi [Dsch2015], [LiOg2011]) è un metodo per risolvere problemi cosiddetti critici (ad.es. mal definiti). Dal punto di vista di RE, il design thinking è una combinazione di tecniche di elicitazione e validazione. Al centro di questo metodo ci sono (a) un team multidisciplinare che lavora sul problema e possiede una vasta gamma di conoscenze necessarie per risolvere il problema; (b) un ambiente di lavoro in cui il team può lavorare sulle idee; e (c) un processo iterativo che si articola nelle seguenti fasi:

- Empatizzare: in questa fase, il team sviluppa empatia per comprendere le persone coinvolte nel problema da risolvere.
- Definire: in questa fase, il team riformula il problema da risolvere per ottenere una comprensione condivisa dei suoi dettagli.
- Ideare: in questa fase, il team si concentra sulla generazione di idee. L'obiettivo qui non è quello di sviluppare un'idea. Il team sviluppa, invece, più idee possibili. Alla fine di questa fase, il team seleziona le idee più promettenti per la prototipazione.
- Prototipazione: in questa fase, il team crea, partendo dalle idee sviluppate, prototipi molto semplici (non necessariamente software!). Il principio applicabile è che il prototipo deve essere il più realistico e il più economico possibile.
- Testing: in questa fase il team testa il prototipo con clienti reali per ottenere un feedback sulle loro idee. Un principio principale per la fase di test è "show not tell", cioè il prototipo dovrebbe essere in grado di parlare da solo in modo che l'utente possa fornire un feedback genuino.

Le fasi del design thinking sono scalabili e possono essere eseguite in progetti che vanno da pochi giorni a diverse settimane. Inoltre le fasi non definiscono una sequenza rigorosa. Ogni volta che è necessario, il team può decidere di tornare indietro o saltare in avanti nel processo. Con questo in mente, il design thinking è in linea con il valore di Agile "rispondere al cambiamento più che seguire un piano" Il risultato finale di un processo di design thinking è un insieme di prototipi che rappresentano soluzioni innovative e validate al problema definito all'inizio. Quindi il design thinking può essere utilizzato per sviluppare idee per software di alto valore e quindi supporta il 1^ principio Agile. Come affermato precedentemente, un team multidisciplinare e l'ambiente di lavoro sono elementi chiave del processo, che risulta così allineato al 5^ principio Agile. Uno dei principali obiettivi della fase di prototipazione è produrre prototipi economici e "leggeri", in linea con il principio di semplicità Agile.

Design Sprint [KnZK2016] è un processo di cinque giorni per lo sviluppo di idee basate sui principi di progettazione, prototipazione e testing con i clienti finali. Dal punto di vista del RE, il Design Sprint è anch'esso una combinazione di tecniche di elicitazione e di validazione. Il fulcro di questo metodo è il modo di lavorare time-boxed; ogni giorno è dedicato a una delle seguenti attività: far emergere le conoscenze del team, abbozzare idee, decidere quali idee prototipare, prototipare le idee selezionate e infine testare le idee con clienti reali. La differenza importante rispetto allo sviluppo Agile è che il prototipo non ha bisogno di essere un software. È importante rendersi conto che il termine "Sprint" non si riferisce allo Sprint Scrum.

Lean Startup [Ries2011] è un approccio per lo sviluppo di startup aziendali e gestionali che è molto ben accettato nella comunità Agile. Da una prospettiva RE, anche Lean Startup contiene numerose idee che sono molto interessanti. Due esempi sono lo speciale approccio allo sviluppo del prodotto e il Prodotto Minimo Utilizzabile. L'approccio di sviluppo del prodotto è chiamato build –measure– learn e si concentra in particolare sull'apprendimento continuo delle esigenze del cliente. Il Prodotto Minimo Utilizzabile (MVP) è "una versione di un nuovo prodotto che consente a un team di raccogliere la massima quantità di conoscenze validate sui clienti con il minimo sforzo".[Ries2011] Un altro concetto importante del Lean Startup è il pivot, "una correzione strutturata del percorso progettata per testare una nuova e fondamentale ipotesi relativa al prodotto, alla strategia e al motore della crescita"[Ries2011]. Da un punto di vista RE, queste idee sono una combinazione di tecniche di elicitazione e di validazione. Invece di elicitare e convalidare i requisiti basati su concetti o documenti, l'elicitazione e la convalida vengono eseguite con il prodotto reale, il che è preferibile secondo Ries in circostanze di estrema incertezza. Ries sottolinea che l'MVP non deve necessariamente essere un software totalmente funzionale e completo. Il libro menziona un esempio in cui una pagina web molto semplice per la vendita di scarpe, viene testata con un processo di spedizione manuale, per validare l'esigenza di un acquisto online di scarpe.

Questi approcci dimostrano che Agile è molto più di Scrum. Gli esempi di Design Thinking, di Design Sprint e di Lean Startup mostrano che in RE@Agile esistono approcci di lavoro concettuale che condividono la visione dei metodi Agile e sono quindi pienamente compatibili con le organizzazioni che desiderano sviluppare software in modo Agile. Questi e altri approcci non dovrebbero essere scambiati con nuove forme dell'approccio waterfall o dell'upfront thinking. Essi possono e devono essere utilizzati nell'ambito dello sviluppo Agile (ad es. all'interno di una o più iterazioni) per progettare aspetti specifici di un sistema. In alternativa essi possono essere utilizzati come attività che precedono lo sviluppo Agile (ad es. una fase veloce dell'upfront thinking). In tal modo questi approcci aiutano a superare il limitato potenziale di innovazione nello sviluppo Agile.

2 Fondamenti di RE@Agile (L2)

Durata: 2.5 ore

Termini: Product Owner, Product Backlog, Sprint Backlog, Epic, User Story, Story Map

Obiettivi Formativi

EO 2.1.1 Conoscenza di esempi dei metodi Agile
EO 2.2.1 Conoscenza di Scrum come esempio: i suoi ruoli, processi, artefatti e la loro rilevanza per il RE
EO 2.2.2 Conoscenza delle responsabilità del Product Owner di Scrum
EO 2.2.3 Conoscere il concetto di product backlog e come si differenzia da un documento di specifica dei requisiti
EO 2.3.1 Capire delle differenze fra un classico Requirements Engineer e un Product Owner Scrum
EO 2.4.1 Comprendere perché l'ingegneria dei requisiti dovrebbe essere parte di un processo continuo
EO 2.5.1 Conoscenza dello sviluppo value-driven – guidato dal valore (ad.es.. assegnare priorità ai requisiti)
EO 2.5.2 Conoscenza del valore per la gestione del rischio e delle opportunità
EO 2.5.3 Conoscenza di esempi di valore in organizzazioni profit e non-profit
EO 2.6.1 Conoscenza di come semplificare il Processo di Definizione del Prodotto e come definire un Prodotto Minimo Utilizzabile
EO 2.7.1 Conoscenza del valore dei processi continui e della loro curva di apprendimento

2.1 Metodi Agile (una panoramica) (L1)

Sono stati sviluppati molti metodi condividendo i valori del manifesto Agile. Questa unità educativa vi fornirà una panoramica di alcuni di essi al fine di illustrare la diversità degli approcci. L'elenco non intende essere esaustivo e presenterà i metodi principalmente dal punto di vista del RE

Crystal è una famiglia di metodi sviluppata da Alistair Cockburn [Cock1998]. Egli suggerisce che ogni progetto necessita di un proprio modello di processo personalizzato. A seconda della dimensione, della complessità e della criticità del progetto Alistair suggerisce ruoli, attività e prodotti del lavoro adeguati. Crystal Clear – il metodo per progetti piccoli e meno critici – è molto simile a XP. Crystal Orange e Crystal Red aggiungono un po' più di formalismo per affrontare progetti più grandi. Dal punto di vista dei requisiti Alistair suggerisce (tra le altre cose) di lavorare con modelli di use case a bassa formalità e mock up.

La story mapping (vedi **Sviluppo Lean** [Popp2003] **e Kanban** si basano su principi utilizzati per la prima volta nella produzione automobilistica negli anni '40. Sono stati adattati per progetti IT nel contesto dei metodi Agile [Ande2012]. Essi cercano di scoprire 7 tipi di sprechi nel processo di produzione (prodotti intermedi non finiti, sovra-produzione, difetti, ...) e di eliminare gradualmente ciascuno di essi per accelerare la consegna finale.

Scrum [Scrum2020] è un framework per sviluppare e sostenere soluzioni di valore come prodotto in ambienti complessi. Il framework si concentra sullo sviluppo iterativo e incrementale basato sull'empirismo e sul pensiero snello. Identifica solo tre ruoli chiave (denominati accountabilities in Scrum [Scrum2020]): un Product Owner (per gestire il product backlog, ossia definire la visione e tutti i requisiti rilevanti di un prodotto), gli sviluppatori che implementano questi requisiti negli Sprint (termine per indicare le iterazioni in Scrum) e uno Scrum Master per guidare e facilitare l'applicazione di Scrum sia per lo Scrum Team che per l'organizzazione in via di sviluppo. Parleremo di Scrum in maggior dettaglio nella prossima sezione.

TDD (Test driven development) [Beck2003] si basa sull'idea di scrivere il test prima di codificare la funzione corrispondente. I casi di test sono una specifica esatta e dettagliata dei requisiti che il prodotto deve soddisfare.

XP (eXtreme programming) [Beck2004]: XP enfatizza la comunicazione diretta tra un cliente e un programmatore ("il cliente in-loco" siede proprio accanto al programmatore, discutendo costantemente dei requisiti e ottenendo un feedback immediato sotto forma di feature implementate).

2.2 Scrum (insieme a buone pratiche) come un esempio (L1)

Scrum è lo schema Agile più popolare e più adottato. Scrum è un framework leggero che aiuta le persone, i team e le organizzazioni a generare valore attraverso soluzioni adattive per problemi complessi. La Guida Scrum [Scrum2020] fornisce una definizione di Scrum e dei suoi componenti essenziali.

In questo metodo, per definizione [Scrum2020], ci sono:

- tre responsabilità/ruoli (Scrum master, product owner, sviluppatori). Questi tre sono anche indicati collettivamente come team Scrum.
- cinque eventi (Sprint, pianificazione dello Sprint, daily scrum, revisione dello Sprint, retrospettiva dello Sprint)
- tre artefatti[1] e i relativi impegni (product backlog, Sprint backlog, increment)

Scrum non raccomanda alcuna pratica di engineering.

Scrum suggerisce di sviluppare prodotti in modo iterativo e incrementale in una serie di Sprint, un'iterazione con un timebox (vincolo temporale) di un mese o meno. Ogni Sprint si traduce in un incremento utilizzabile, che rappresenta un passo concreto verso l'obiettivo del prodotto. In questo caso, è il proprietario del prodotto a decidere se consegnare o meno l'incremento al cliente, con la direttiva di solito di farlo, tranne che per motivi significativi (ad esempio, rischi potenziali).

[1] Prodotti di lavoro secondo la terminologia CPRE [Glinz2022]

Lo Sprint

Lo Sprint è il driver essenziale per lo sviluppo in quanto è un processo iterativo di plan –do– check –act che consente brevi cicli di feedback. Ogni Sprint inizia con lo Sprint Planning, un evento in cui il Team Scrum collabora per definire cosa può essere consegnato nel prossimo Incremento di Prodotto e come ottenerlo (decomposizione). L'origine di questo piano è tratta dal Product Backlog (un elenco ordinato e dinamico di tutti i requisiti attualmente noti per il prodotto). A differenza del documento di specifica dei requisiti dell'ingegneria dei requisiti classica, il Product Backlog non è ancora una raccolta completamente formulata di tutti i requisiti. Spesso (soprattutto all'inizio dello sviluppo di un prodotto) si tratta solo di un elenco di idee formulate in modo approssimativo che si sviluppano nel tempo e diventano requisiti concreti. Quanto più un elemento del Product Backlog è prioritario e quanto più si avvicina il momento dell'implementazione (ad esempio, in uno dei due Sprint successivi), tanto più precisa è la descrizione di un elemento del Product Backlog. Il Product Owner è responsabile del Product Backlog, che nella sua struttura attuale è guidato dall'attuale Product Goal. Per maggiori dettagli, vedere 3.1.1 Documenti di specifica vs. Product Backlog.

La *pianificazione dello Sprint (Sprint Planning),* che viene eseguita per selezionare gli elementi che saranno implementati nello Sprint successivo, porta ai seguenti risultati: lo Sprint Goal (Why) e lo Sprint Backlog (What, vale a dire la selezione degli elementi e loro scomposizione per lo Sprint) e lo Sprint Plan (come raggiungere l'obiettivo).

Successivamente, il *Team Scrum* inizia ad implementare *l'Incremento di Prodotto*. I developers sono responsabili della conversione degli elementi selezionati in risultati concreti. Lavorano con il *Product Owner* per perfezionare le voci del *Product Backlog* e condividono il feedback sull'attuale implementazione. I developers si sincronizzano ogni giorno durante il Daily Scrum per valutare i progressi verso l'obiettivo di Sprint e aggiornare lo Sprint Backlog dove necessario.

Il lavoro degli sviluppatori è guidato dalla "Definizione di Done": una definizione di ciò che deve essere ottenuto affinché un *incremento* sia completo. La "Definizione di Done" aiuta gli stakeholder a comprendere in modo inequivocabile il progresso del prodotto.

Quando lo Sprint timebox è scaduto (un mese al massimo), l'Incremento di Prodotto viene ispezionato nella Sprint Review dal Team Scrum e dai principali stakeholder per valutare i risultati dello Sprint e discutere le prossime cose che si potrebbero fare. Il Product Backlog viene aggiornato conseguentemente.

Lo Sprint si conclude con la *Sprint Retrospective*, durante la quale lo Scrum Team esamina come può diventare più efficiente, cosa funziona bene e cosa potrebbe essere migliorato. Appena terminata la *Retrospettiva di Sprint*, lo Sprint è terminato e inizia lo Sprint successivo.

Buone pratiche

Anche se Scrum non prevede tecniche di RE, la comunità Agile ha sviluppato alcune buone pratiche che si adattano bene a Scrum.

La Guida Scrum utilizza il termine Product Backlog Items per gli elementi elencati nel Product Backlog. Questo è un termine generico per identificare qualsiasi tipo di informazione

sul prodotto da sviluppare, ma molti Product Backlog Item sono effettivamente requisiti o possono essere affinati per diventare requisiti.

La comunità Agile propone di:

- Scomporre i requisiti in: Epic, opzionalmente Feature e User Story (livello di granularità), vedere [CPREALAGILE2022].
- Distinguere tra Requisiti Funzionali (capacità), Requisiti di Qualità (comportamento) e Vincoli (ambiente)
- Raggruppare i requisiti per Tema

Una buona pratica (aggiuntiva) è stata sviluppata per descrivere le caratteristiche del Product Backlog: il backlog dovrebbe essere "DEEP" (Detailed appropriately, Estimated, Emergent ,Prioritized).[Cohn2004] All'interno dello Sprint c'è il grooming e l'affinamento del backlog, che è una parte essenziale del processo di sviluppo di Agile.

Con la "Definition of Done" (DoD) il team sviluppa una comprensione comune di quando un elemento del Product Backlog è completo e pronto a diventare parte dell'incremento utilizzabile [Scrum2020].

Un'altra buona pratica suggerisce di applicare la regola INVEST.[Wake2003] L'acronimo copre i seguenti criteri:

- **I**: Indipenedente l'uno dall'altro
- **N**: Negoziabile
- **V**: di Valore
- **E**: Estimabile
- **S**: Abbastanza piccolo (Small) per stare in uno Sprint
- **T**: Testabile

RE offre criteri corrispondenti per i buoni requisiti (vedi [CPREFL2022]):

- concordato
- non ambiguo
- necessario
- consistente
- verificabile
- fattibile
- tracciabile
- completo
- comprensibile

2.3 Differenze e elementi comuni tra i Requirements Engineer e i Product Owners (L2)

Dopo aver discusso i principi dello schema Scrum, confrontiamo il ruolo del Requirements Engineer tradizionale con il ruolo del Product Owner. Va detto che il ruolo del Product Owner è stato nel frattempo adottato da molti approcci Agile (anche al di fuori di Scrum). Pertanto, la seguente descrizione non deve essere necessariamente intesa solo per Scrum.

Le attività principali di un Requirements Engineer sono [CPREFL2022]:

- Elicitazione dei requisiti
- Documentazione dei requisiti
- Validazione e negoziazione dei requisiti
- Requirements management

Come spiegato precedentemente, le responsabilità principali di un Product Owner sono:

- Assicurare che il team rilasci un valore (di business) costantemente: ciò significa che il Product Owner deve bilanciare la visione a lungo termine del prodotto (product goal, product vision) con le esigenze a breve termine; deve assegnare le priorità nel Product Backlog coerentemente con dei criteri definiti e deve passare in rassegna i risultati del team insieme agli stakeholder alla fine di ogni iterazione (Sprint).
- Gestire tutti gli stakeholder: il Product Owner è responsabile per l'inoltro al team. di requisiti coerenti. Deve raccogliere i requisiti di tutti gli stakeholder e assicurarsi che non si contraddicano a vicenda. Qualsiasi controversia tra gli stakeholder deve essere risolta al fine di lasciare liberi gli sviluppatori da eventuali disaccordi.
- Fornire continuamente al team gli elementi del backlog che hanno il maggior valore (di business): La granularità di questi requisiti deve essere abbastanza piccola da rientrare in un'iterazione (Sprint). Per eventuali domande che sorgano dopo la riunione di Planning, il Product Owner deve essere disponibile per chiarire rapidamente.

Confrontando i due ruoli, risulta che sia i Requirements Engineer sia i Product Owner (insieme a tutti gli altri stakeholder) devono svolgere i compiti chiave di elicitazione, documentazione, validazione e gestione dei requisiti. Le notazioni e gli strumenti usati, tuttavia, sono solitamente meno formali in ambienti Agile:

- per esempio, story card invece di documenti dei requisiti
- più conversazione e meno scrittura
- maggiore enfasi sullo stato attuale dei requisiti, minor enfasi su versionamento e cronologia

Mentre continua a mantenere una responsabilità globale sui requisiti di alta qualità, il ruolo di Product Owner è più ampio di quello di un tradizionale Requirements Engineer, in quanto è responsabile del successo del prodotto nel suo complesso, raccogliendo continuamente feedback dal business e aggiornando e prioritizzando di conseguenza il backlog.

Per quanto riguarda le attività di ingegneria dei requisiti, si può riassumere che il Product Owner ne risponde. Può essere coadiuvato da persone con esperienza nell'Ingegneria dei requisiti o svolgere personalmente questi compiti. Tuttavia, la responsabilità del risultato dell'Ingegneria dei requisiti e del Product Backlog rimane in capo al Product Owner.

2.4 Requirements Engineering come processo continuo (L2)

In Agile, quindi, il RE è meno una fase distinta dello sviluppo e più un'attività iterativa e continua. Non ha l'obiettivo di avere tutti i requisiti elicitati e analizzati prima che possa iniziare la progettazione e l'implementazione; i requisiti e i prodotti vengono creati sia iterativamente che in modo incrementale.

Il RE è perciò un'attività continua che dura quanto la vita del prodotto stesso. Ciononostante, questo processo ha risultati intermedi ben definiti: i requisiti previsti che promettono il massimo valore (di business) secondo i criteri definiti, dovrebbero essere "ready per l'implementazione" (nel senso della "definizione di ready" sopra descritta). Altri requisiti meno urgenti saranno rifiniti solo una volta completati quelli urgenti.

Il "Processo continuo" non esclude alcune attività preliminari importanti. Anche se i requisiti vengono chiariti sulla base del "need to know", vi sono alcuni aspetti dei requisiti che dovrebbero comunque essere affrontati anticipatamente nel ciclo di vita. Esempi di ciò sono: definizioni di visioni o obiettivi, conoscenza degli stakeholder e determinazione dello scopo del prodotto. L'avvio dell'implementazione senza tali attività aumenta considerevolmente il livello di rischio.

2.5 Sviluppo value driven (L1)

I metodi Agile si sforzano di fornire continuamente valore (di business) all'utente finale. Spesso il valore può essere espresso direttamente in termini finanziari, in una maggiore quota di mercato o in termini di soddisfazione del cliente. Questo approccio viene utilizzato frequentemente nelle organizzazioni guidate dal profitto, mentre è meno chiaro come definire il valore per le organizzazioni non-profit. In queste ultime potrebbero essere più rilevanti misure come l'indice di utilizzo o l'indice di felicità relative a un prodotto (ad es. numero di clic su siti Web o donazioni a un'organizzazione non -profit).

Un tipo di valore diverso è la riduzione del rischio. Buoni approcci Agile cercano di bilanciare il valore di business e la riduzione del rischio in relazione alle iterazioni.

Per determinare quali requisiti portino al valore ottimale, i metodi Agile spesso vanno alla ricerca di prodotti minimi utilizzabili (MVP) o di prodotti minimi commerciabili (MMP) come spiegato nella prossima parte.

2.6 Semplicità come concetto essenziale (L1)

In un mondo complesso, la semplicità è un modo per affrontare la complessità mediante un processo in cui si cerca di:

- Creare una risposta semplice anche se potenzialmente incompleta a un problema, creando così un piccolo incremento di valore;
- Acquisire la capacità di apprendere di più sul contesto, basandosi sull'esperienza del mondo reale;
- Adattare e iterare la creazione di valore e il rilascio a un ritmo sostenibile;

- Risparmiare risorse da un'idea non redditizia per riallocarle su una nuova idea, e imparando rapidamente dagli errori commessi.

La semplicità è in qualche modo opposta alla "perfezione". Ciò detto, la semplicità non significa "scarsa qualità" ma piuttosto "portata minima" o "servizio minimo" – ma sempre con alta qualità. La qualità non è negoziabile.

Il Manifesto Agile descrive la semplicità come "l'arte di massimizzare la quantità di lavoro non svolto". Questo non significa evitare qualsiasi sforzo per ridurre il carico di lavoro, ma piuttosto essere chiari sul lavoro da svolgere, creare valore e raggiungere obiettivi specifici. Se il team realizzasse requisiti non allineati, allora massimizzerebbe il lavoro non valido, che dovrebbe essere evitato per non bruciare costi e tempo senza un motivo valido.

Spesso si definiscono due tipi di prodotti "semplici": MVP e MMP

Il Minimum Viable Product, o Prodotto Minimo Utilizzabile (MVP) è un concetto dell'avvio snello (vedi [Ries2011]) ed è definito come il prodotto più piccolo in grado di creare un'esperienza per l'utente finale e fornire feedback al team. Tale feedback è un input importante per far evolvere il prodotto. Molte start-up adottano questo modo di lavorare, poiché esso consente di creare un rapido ritorno dell'investimento con un basso livello di rischio.

L'obiettivo dei Minimum Marketable Products o Prodotti Minimi Commercializzabili (MMP) prevede un ulteriore passo avanti. Il problema non è solo fornire un feedback tempestivo, indirizzando così i successivi passi dei requisiti, ma anche creare immediatamente valore. Molti prodotti possono essere utilizzati in una semplice "versione 1" semplice senza avere ancora tutte le caratteristiche e le qualità desiderate, creando così entrate economiche per finanziare il miglioramento continuo del prodotto. In questo caso, spesso si utilizza il processo Lean Startup (Costruire, Misurare, Imparare) in modo iterativo.

2.7 Ispezionare e adattare (L1)

I metodi Agile sottolineano l'importanza di riscontri frequenti e veloci. Dopo ogni iterazione (a volte anche più spesso) il team dovrebbe discutere se il processo di sviluppo stia funzionando e se dovrebbe essere migliorato.

Nel processo di feedback, tutti dovrebbero inizialmente esaminare l'attuale processo di sviluppo. Il team dovrebbe esaminare criticamente i metodi usati, gli strumenti, la cooperazione nel team, ecc. A tutti viene chiesto di rispondere a domande quali: Che cosa ha funzionato bene? Che cosa non ha funzionato bene? Cosa dovremmo provare nella prossima iterazione?

È importante che il prodotto o la soluzione vengano revisionati alla fine di ogni iterazione. Il team dovrebbe valutare la velocità del proprio lavoro e modificare o regolare la velocità di implementazione pianificata per l'iterazione successiva.

Questo vale anche per il processo dei requisiti. Le conseguenze di questi approfondimenti dovrebbero portare all'adattamento a breve termine delle fasi di miglioramento del processo.

3 Prodotti e tecniche di lavoro in RE@Agile (L2)

Durata: 2.5 ore

Termini: Visione di Prodotto, Roadmap di Prodotto, Product Backlog, Sprint Backlog, User Story, Criterio di Accettazione, Feature, Requisito Funzionale, Requisito di Qualità, Epic, Modello di Contesto, Story Map, Definizione di Ready, Definizione di Done

Obiettivi Formativi

EO 3.1.1 Conoscenza della differenza tra documenti di specifica tradizionali e backlog
EO 3.1.2 Conoscenza del valore delle visioni e degli obiettivi
EO 3.1.3 Conoscenza del valore aggiunto per l'uso di context models in RE
EO 3.1.4 Conoscenza di come distinguere i tre tipi di requisiti
EO 3.1.5 Comprensione dei diversi livelli di granularità nei requisiti
EO 3.1.6 Comprensione dei diversi formati delle specifiche per i diversi tipi di prodotti del lavoro nei processi Agile (per es. testuale piuttosto che basato su modelli o su diagrammi)
EO 3.1.7 Comprensione del valore di Termini, Glossari e modelli Informativi
EO 3.1.8 Comprensione delle specifiche dei Requisiti di Qualità e dei Vincoli nei processi di Requirements Engineering Agile
EO 3.1.9 Conoscenza dei Criteri di Accettazione e Adattamento
EO 3.1.10 Comprensione dell'uso della Definizione di Ready e della Definizione di Done nei processi di Requirements Engineering Agile
EO 3.1.11 Conoscenza della differenza tra Prototipo, Incremento e Spike
EO 3.1.12 Conoscenza dei diversi tipi di prodotti di lavoro nei Processi RE@Agile (Context Model, Epic, User Story, Backlog, Roadmap, Requisito, Definizione di Done, Definizione di Ready)
EO 3.2.1 Comprensione di come elicitare i requisiti nei processi di Requirements Engineering Agile
EO 3.2.2 Conoscenza di come creare e mantenere i backlog nei processi di RE Agile
EO 3.2.3 Saper validare e negoziare i requisiti in RE@Agile
EO 3.2.4 Conoscenza di come gestire i requisiti in RE@AGILE

3.1 Prodotti di lavoro in RE@Agile (L2)

3.1.1 Documenti di specifica vs. Product Backlog

Per creare prodotti o soluzioni dai requisiti, i requisiti stessi non possono essere isolati, ma devono essere organizzati e documentati in un elenco ordinato di tutto ciò che potrebbe essere necessario nel prodotto.[Scrum2020] Indipendentemente da qualsiasi metodologia, l'elenco dei requisiti è considerato il prodotto di lavoro principale per i Requirements Engineer, gli analisti di business o per i Product Owner o chiunque sia responsabile per l'analisi dei requisiti. Metodi diversi suggeriscono per l'elenco dei requisiti forme e nomi diversi.

Il Requirements Engineering di solito chiama questo prodotto di lavoro, che è il risultato del processo di elicitazione, Specifica dei Requisiti Utente, Specifica dei Requisiti di Sistema o Specifica dei Requisiti Software, a seconda di chi lo scrive e del livello di dettaglio che copre. Non si tratta necessariamente di un documento, ma di una raccolta di requisiti in una qualsiasi forma fisica (carta, repository, database ...).

I metodi Agile, in particolare Scrum, definiscono il termine Product Backlog per indicare la raccolta generale dei requisiti (e altre informazioni relative al prodotto, vedi 2.1) da implementare in futuro e Sprint Backlog per indicare quei requisiti che sono stati selezionati per l'iterazione successiva (sprint in Scrum) [Scrum2020]. Anche in questo caso, la forma fisica dell'arretrato non ha importanza. Il backlog potrebbe consistere di schede indicizzate o note appese al muro oppure essere inglobato in uno strumento software appropriato. Sebbene i nomi e il trattamento possano differire, tutti i prodotti di lavoro seguono le stesse idee e offrono una base per l'elicitazione, la documentazione, la negoziazione, la validazione e la gestione dei requisiti.

In 2.2 abbiamo imparato che il DEEP è una buona pratica, in modo che il backlog dei prodotti sia dettagliato in modo appropriato, stimato, emergente e prioritario. Queste caratteristiche sono importanti per il backlog di prodotto e sono collegate o dipendenti l'una dall'altra. Per il Product Owner, in quanto "ottimizzatore di valore", la classifica o l'ordine del backlog di prodotto è il più prezioso e importante da raggiungere. La stima supporta questo aspetto e i dettagli appropriati aiutano a concentrarsi sulle parti importanti, in modo che siano di supporto all'ordine. Indipendentemente dal modo in cui vengono documentati i requisiti, alcuni elementi devono essere assolutamente catturati. Essi includono vari tipi di requisiti: obiettivi e visioni, definizione dell'ambito del sistema o del prodotto, requisiti funzionali, requisiti di qualità e vincoli e un glossario (con le definizioni di termini e abbreviazioni pertinenti). Come illustrato più avanti, gli approcci per le specifiche dei requisiti possono variare nelle notazioni, nella sintassi e nel livello di dettaglio. Non avere alcuna specifica (cioè fidarsi solo della comunicazione verbale tra gli stakeholder senza alcun requisito scritto) non è in genere un'alternativa, poiché i documenti scritti sono spesso la base per la negoziazione, i test di accettazione, gli scopi legali e altro ancora. Più tutti gli stakeholder comunicano tra loro, minore è la necessità di scrivere, ma i risultati, i requisiti, devono comunque essere catturati in una forma nota (testuale o grafica). I diversi aspetti della raccolta dei requisiti generali saranno discusse in dettaglio nei paragrafi seguenti.

3.1.2 Visione e obiettivi

Ogni processo di sviluppo dovrebbe essere guidato da visioni e obiettivi che definiscano le caratteristiche del prodotto che una volta, dimostrerebbero che la soluzione adottata è positiva. Identificare visioni o obiettivi, concordati con tutti gli stakeholder rilevanti nel più breve tempo possibile, è della massima importanza per qualsiasi attività relativa ai requisiti del sistema o prodotto in esame.

Nei processi di sviluppo Agile il termine "visione di prodotto" è comunemente usato per sottolineare che ogni risultato del processo di sviluppo dovrebbe avere un valore riconoscibile in relazione alla vision di prodotto.

Per definire questo valore, può essere necessario innanzitutto definire chiaramente quali sono i valori che l'azienda si prefigge di raggiungere.

Gli obiettivi dei diversi stakeholder possono essere conflittuali, il che significa che essi devono negoziare per ottenere una visione o una serie di obiettivi concordati. In alternativa, la presenza di requisiti in conflitto potrebbe indicare che sono necessarie varianti del prodotto (ad es. un iPhone piccolo e uno grande) o addirittura prodotti diversi (un iPhone e un iPad).

Lo sviluppo Agile utilizza spesso obiettivi con granularità diversa, quali obiettivi per orizzonti temporali o intervalli di pianificazione diversi. Ad esempio, potrebbero esserci obiettivi ad un anno per consentire negoziazioni sulle consegne (tempi e contenuti), obiettivi a tre mesi per la pianificazione dei rilasci e obiettivi di iterazione/sprint per l'iterazione successiva [High2009].

Visioni di prodotto a lungo termine e obiettivi a breve termine sono utili per sottolineare i risultati più importanti da raggiungere in un determinato periodo di tempo e per contribuire ad allineare tutti gli stakeholder su una "missione comune". Tutti gli obiettivi possono essere rappresentati efficacemente in una tabella temporale all'interno della roadmap.

La visione o gli obiettivi di un prodotto costituiscono la forma di requisiti più astratta e non possono essere portati allo sviluppo senza ulteriori perfezionamenti. Essi forniscono una guida facile e comprensibile per l'intero processo di sviluppo Agile. Ogni requisito dovrebbe essere verificato rispetto agli obiettivi per accertare il contributo del requisito ai diversi obiettivi. Un requisito senza una relazione con l'insieme degli obiettivi può essere un indicatore di assenza di valore. Di conseguenza, la dichiarazione di vision di prodotto e gli obiettivi concordati dagli stakeholder sono prodotti di lavoro molto importanti per il successo dei processi di sviluppo Agile, perché stabiliscono il quadro di riferimento per tutte le attività di sviluppo senza limitare inutilmente la creatività degli sviluppatori.

3.1.3 Modello del contesto

La dichiarazione di visione insieme agli obiettivi degli stakeholder specificano le richieste complessive che il sistema o il prodotto dovrebbe soddisfare per raggiungere il suo scopo. I context models rappresentano invece un diverso punto di vista, poiché il loro scopo è descrivere proprietà particolari dell'ambiente (contesto) in cui opereranno il sistema o il prodotto.

I requisiti del sistema o del prodotto sono spesso specificati in funzione di ipotesi sull'ambiente. I context models sono un modo strutturato per documentare ipotesi rilevanti sul contesto. È utile rendere esplicite tali ipotesi al fine di stabilire una visione condivisa e concordata sull'ambiente operativo del sistema o del prodotto per l'intero team e per altri stakeholder rilevanti.

In caso di incertezza, i requisiti specificati saranno modificati solo se le ipotesi documentate nei context models si dimostreranno vere, dimostrando che i context models rappresentano correttamente il contesto operativo effettivo del sistema o del prodotto.

I context models sono un prodotto di lavoro efficace, poiché distinguono chiaramente tra il sistema o prodotto da sviluppare e il suo contesto, costituito, ad esempio, da sistemi adiacenti e utenti umani (vedi [CPREFL2022]). Grazie a questa differenziazione, è possibile assegnare le funzionalità.

Ciò consente di distinguere tra le responsabilità del sistema o del prodotto stesso e le responsabilità dei sistemi adiacenti o degli utenti umani nel contesto, che collaborano durante un'operazione per realizzare la visione complessiva. Pertanto i context models possono essere utilizzati per chiarire e specificare le interfacce esterne del sistema o del prodotto da sviluppare.

Questi modelli possono essere documentati utilizzando formati diversi, come context diagrams proposti per l'analisi di sistemi strutturati, diagrammi di use case, diagrammi di definizione di blocchi SysML, UML component diagrams o UML class diagrams. Qualsiasi notazione è adatta se distingue chiaramente nell'ambiente tra il sistema o il prodotto da sviluppare e le interfacce esterne verso persone o altri sistemi. Deve inoltre documentare con un livello di dettaglio adeguato le relazioni tra questi elementi e il sistema o prodotto da sviluppare. Anche semplici diagrammi disegnati a mano con linee e figure geometriche possono essere pragmatici e utili.

Indipendentemente dalla forma della documentazione, un context model è un prodotto di lavoro molto prezioso ed è raccomandato durante un processo di sviluppo Agile. Esso delimita l'area (all'interno dell'ambito) in cui gli analisti sono liberi di prendere decisioni, mentre le interfacce esterne (vale a dire il confine tra ambito e contesto) devono essere negoziate con i sistemi adiacenti.

3.1.4 Requisiti

I requisiti devono quindi essere acquisiti in base alla visione e agli obiettivi e delimitati dal modello di contesto. In genere si distingue tra tre tipi di requisiti: requisiti funzionali, requisiti di qualità e vincoli [CPREFL2022].

3.1.5 Granularità dei Requisiti

Gli stakeholder spesso comunicano le loro esigenze a diversi livelli di granularità: si va da requisiti grossolani, quali obiettivi aziendali generali, a requisiti particolareggiati, che specificano i dettagli delle funzionalità del sistema previsto. I requisiti funzionali e di qualità (vedi 3.1.8) possono (e dovrebbero!), perciò essere discussi e documentati a diversi livelli di astrazione.

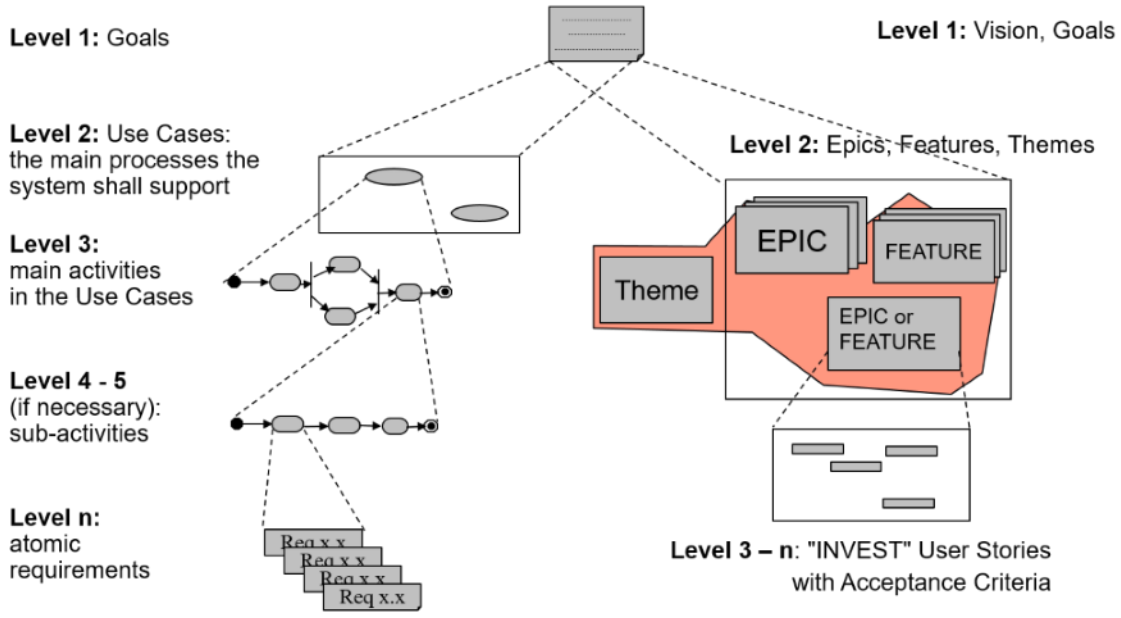

Figura 1: Granularità dei Requisiti: scomposizione del Caso d'Uso e varianti Agile, Casi d'Uso, specifiche Caso d'Uso e activity diagrams

Un approccio tipico delle metodologie non-Agile è rappresentato sul lato sinistro della Figura 1. In questo approccio, i casi d'uso vengono utilizzati prima di tutto per affinare visioni e obiettivi e fornire una scomposizione delle funzionalità del sistema previsto (Vedi Figura 1 – Livello 2). L'immagine mostra solo un esempio per un gruppo di requisiti di alto livello. Altri esempi includono il raggruppamento per feature, oggetti di business, flussi di dati o componenti di soluzioni esistenti.

Al livello di granularità successivo, ogni use case viene elaborato per descrivere i passi necessari per svolgere la funzione rappresentata. Sono qui disponibili diverse alternative in base al livello di complessità:

1. Descrizione verbale (Testo semplice)
2. Template di Use Case (formato semi-strutturato o flusso narrativo)
3. UML Activity diagram (o qualsiasi altro tipo di modello funzionale grafico come Data Flow Diagrams, BPMN, Context-Diagram, Diagramma di Stato, Business Object Model, ecc.)

Il livello 3 nella Figura 1 mostra un esempio di use case elaborato con diagrammi di attività UML.

A seconda della complessità del problema, possono essere forniti ulteriori livelli di granularità, ad es. con la decomposizione di attività in sotto-attività (vedi Livello 4 in Figura 1) o con specifiche testuali dei requisiti per le singole attività (vedi Livello 5 in Figura 1).

Requisiti cliente poco dettagliati possono quindi essere perfezionati in specifiche più dettagliate della funzionalità prevista.

Occorre sottolineare che l'approccio descritto sopra implica un processo di analisi e scomposizione dei requisiti di tipo top-down (dall'alto in basso). Questa visione un po' idealizzata non è sempre realistica. I requisiti possono infatti essere espressi per la prima

volta a livello di soluzione, con obiettivi più generali stabiliti solo successivamente. Il processo nel mondo reale può quindi essere dall'alto al basso, dal basso all'alto o una loro combinazione.

Il punto chiave è che esistono metodi e tecniche per supportare queste discussioni ai loro diversi livelli di astrazione e per stabilire una gerarchia significativa di requisiti man mano che la conoscenza generale dei requisiti stessi aumenta.

Un'opzione che può essere adottata per descrivere casi d'uso a bassa complessità, è quella di usare solo poche frasi per chiarire i passi necessari da eseguire. Per i casi d'uso di media complessità, può essere una buona scelta adottare altre soluzioni, poiché i modelli dei casi d'uso consentono di descrivere come sequenza di passi di un processo gli use case più complessi. Questo formato semi-strutturato aiuta a comprendere i requisiti descritti in uno use case con una sequenza chiara. Tuttavia lo use case model (modello dei casi d'uso) presenta limitazioni se ci sono attività parallele e situazioni ad alta complessità con scenari multipli.

Infine, gli activity diagrams consentono al Requirements Engineer di utilizzare un formato grafico in cui possono essere visualizzati più scenari. Il diagramma può rappresentare più flussi paralleli e flussi sequenziali di passi. Questa forma di descrizione di uno use case consente di illustrare flussi di use case complessi in un diagramma.

Un problema che porta sempre a discussioni è quando un requisito è descritto in modo sufficientemente dettagliato. L'attenzione principale è rivolta alla collaborazione all'interno del team. Un requisito è descritto in modo sufficientemente dettagliato quando il team (in particolare le persone che implementano il requisito, cioè gli sviluppatori) ha capito cosa si vuole ottenere. Si raccomanda di definire in anticipo criteri chiari in merito. Vedere anche Definizioni di Ready e Don.

Terminologia Agile: Epic, Temi, Feature e User-Story

Sul lato destro della Figura 1 viene mostrato un approccio equivalente usando termini Agile come Epic, Temi, Feature e User Story.

Obiettivi aziendali di alto livello e funzionalità complesse sono catturati come epic o feature e raggruppati per tema. Tali argomenti possono essere sufficientemente complessi da richiedere ad esempio ad un team Scrum più di uno sprint per essere sviluppati [Griff2015].

Argomenti complessi vengono quindi scomposti in user story dettagliate. I criteri per "buone" user story sono stati discussi in precedenza (2.2). Essi dovrebbero rispettare la "definizione di ready" [Kron2008] , soddisfacendo per es. i criteri INVEST [Wake2003]. Soprattutto la "E", la "S" e la "T" sono importanti: Buone user story dovrebbero essere stimabili, abbastanza ridotte da poter essere inserite in un'unica iterazione e testabili.

Ancora una volta, che un progetto segua un processo rigorosamente top-down di affinamento di epic in feature e in user story, o che le user story vengano prima raccolte individualmente con temi comuni ed epic che emergono più tardi bottom-up, la scelta dell'approccio dipenderà dalla natura del progetto e dagli stakeholder coinvolti. In entrambi i casi, Agile fornisce prodotti di lavoro per discutere e dare la priorità sia al quadro generale che ai requisiti pronti per lo sviluppo.

3.1.6 Modelli Grafici e Descrizioni Testuali

Indipendentemente dalla granularità, possono essere scelte notazioni diverse per rappresentare i requisiti funzionali. Essi possono essere espressi scrivendoli in forma di testo semi-strutturato (secondo un modello come le user story) o descritti in un linguaggio naturale o formale (come Gherkin) indicando chiaramente che cosa dovrebbe fare la soluzione.

Poiché è noto che il linguaggio naturale a volte non è preciso e non ambiguo come sarebbe necessario, sono state sviluppate molte notazioni grafiche per superare questa ambiguità o mostrare le interdipendenze, come quelle tra dati e processo. Esempi di queste notazioni sono gli UML activity diagrams (diagrammi delle attività UML), i diagrammi BPMN, i diagrammi di flusso, i diagrammi di stato, i sequence diagrams (diagrammi di sequenza), ecc

La maggior parte di queste notazioni grafiche rappresentano aspetti diversi. Gli activity diagrams e i diagrammi BPMN sono particolarmente adatti per processi abbastanza lineari che mostrano passi, alternative o cicli. I diagrammi di stato invece sono ideali ogni volta che eventi asincroni possono influenzare un flusso. I sequence diagrams sono adatti per "specifiche con esempi" poiché mostrano scenari di interazione concreti senza cercare di essere completi.

Nella modellazione di processi con dati o di dati con interazioni o di processi con interfacce ci sono sia aspetti positivi che negativi. Essi sono interconnessi (i dati supportano il processo) quindi non ci può mai essere una situazione che suggerisca di utilizzare solo un modello o un altro. Un tale approccio sarebbe alquanto controproducente. Se i requisiti testuali sono più facili da comprendere per la maggior parte degli stakeholders, essi possono anche essere frequentemente interpretati erroneamente o fraintesi. I modelli grafici sono più formali e permettono così di evitare interpretazioni differenti o incomprensioni. La scelta dello stile dovrebbe essere determinata dagli obiettivi chiave di RE, garantendo da una parte una comprensione comune a tutti gli stakeholder e fornendo dall'altra una protezione sufficiente contro l'incompletezza e l'incomprensione [GoAk2003].

3.1.7 Definizione di Termini, Glossari e Modelli Informativi

I requisiti funzionali sono spesso incompleti senza una chiara comprensione di tutti i termini utilizzati nelle frasi e nei modelli grafici.

Di conseguenza, una raccolta di tutti i termini di business pertinenti utilizzati in qualsiasi prodotto del lavoro sui requisiti è essa stessa un prodotto del lavoro necessario, che tuttavia viene facilmente trascurato quando ci si concentra solo sui requisiti di business.

La forma minima di questo prodotto del lavoro è un elenco dei termini di business e delle abbreviazioni (descritti testualmente), in genere ordinati alfabeticamente per una ricerca più semplice. Questo elenco a volte viene chiamato dizionario, glossario o elenco di definizioni.

Quando il business è molto complesso, il glossario può essere strutturato raggruppando termini semplici in classi (o entità) e mostrando la panoramica di tutti i termini in formato grafico. I prodotti del lavoro così ottenuti vengono quindi denominati modelli dei dati, modelli informativi, entity-relationship-diagrams (diagrammi entità-relazioni), o UML class diagrams (diagrammi delle classi UML). Oltre alla definizione di termini, questi modelli includono anche associazioni rilevanti tra le entità, cioè le relazioni statiche tra i termini.

Come accennato in precedenza, il Product Backlog è spesso costituito da epics, features e user story, in modo tale da mettere in maggiore evidenza le funzionalità necessarie e in minore evidenza le definizioni dei termini di business. Tuttavia, anche negli approcci Agile, è necessaria una chiara comprensione dei termini di business per assicurare una comprensione condivisa e allineata dei termini usati nei prodotti del lavoro, come epic, features e user story.

3.1.8 Requisiti di Qualità e Vincoli

Oltre ai requisiti funzionali (che specificano le funzioni che un sistema o un prodotto deve fornire), i requisiti di qualità e i vincoli sono di importanza cruciale per il successo del sistema o del prodotto da sviluppare. Tradizionalmente, i requisiti di qualità e i vincoli sono inclusi sotto il termine "requisiti non-funzionali"[CNYM2000]. Gli esperti di ingegneria dei requisiti affermano da decenni l'importanza di questi requisiti "non funzionali". Anche se il termine "requisiti non funzionali" è ancora spesso usato nella pratica, come termine ombrello per i requisiti e i vincoli di qualità, l'IREB usa le categorie più concrete e precise di "requisiti di qualità" e "vincoli", secondo [Glinz2022].

I requisiti di qualità riguardano le qualità specifiche che un sistema o un prodotto deve esporre, ad esempio, in termini di prestazioni, affidabilità, sicurezza, sicurezza fisica o usabilità [ISO25010]. Con un'enfasi sulle esigenze funzionali dei clienti sotto forma di user story, c'è il rischio con Agile che i requisiti di qualità non siano esplicitamente indicati. I requisiti di qualità non possono essere definiti come user story che possono a loro volta essere sviluppate all'interno di un'iterazione: essi descrivono piuttosto un attributo emergente del prodotto in fase di sviluppo e devono quindi essere testati continuamente per tutte le user story. A tale scopo possono essere utili le checklist degli aspetti di qualità di RE (vedere [CPREFL2022]). Un'altra opzione potrebbe essere quella di includere tutti i requisiti di qualità nella Definizione di Fatto (DoD), assicurando che tutti i requisiti di qualità siano soddisfatti in modo che un elemento del backlog diventi parte dell'incremento potenzialmente distribuibile [CPREALAGILE2022].

I requisiti di qualità sono notoriamente difficili da soddisfare nei sistemi esistenti tramite il refactoring, quindi è molto più utile considerare tali aspetti nelle prime fasi del processo.

I vincoli definiscono le restrizioni generali allo spazio di soluzione del sistema o del prodotto da sviluppare [Glinz2022]. I vincoli si presentano sotto diverse forme: vincoli organizzativi (limiti di budget, tempi ristretti, un processo di sviluppo obbligatorio, ecc.), vincoli tecnici (richiedenti un determinato Data Base, l'uso di un linguaggio di programmazione specifico, strutture selezionate, ecc.) e vincoli dell'ambiente all'interno del quale il sistema opererà (standard, norme, regolamenti, ecc.).

Come per i requisiti di qualità, l'identificare troppo tardi i vincoli chiave può essere molto costoso, dal momento che molti di questi aspetti non possono essere aggiunti in modo incrementale. Le decisioni di pianificazione e progettazione dipendono da una buona comprensione di questi problemi, e quindi anche in questo caso è cruciale che i vincoli chiave vengano identificati nelle prime fasi del processo.

Similmente ai requisiti funzionali, i requisiti di qualità e i vincoli possono esistere a diversi livelli di astrazione e devono essere documentati nel Product Backlog, resi visibili al team e testati in ogni iterazione. Pertanto potrebbe essere utile aggiungere i vincoli alla "Definizione di Done" e implementare la loro validazione sotto forma di test di regressione automatizzati.

3.1.9 Criteri di Accettazione e Criteri di Adattamento

Tutti i tipi di requisiti hann scarso valore se il loro soddisfacimento non può essere validato, verificato e testato. Perciò ciascun requisito richiede una serie di criteri che possano essere testati per verificare se il requisito è stato effettivamente soddisfatto. Tali criteri aiutano inoltre la comprensione (e incoraggiano il feedback) descrivendo in termini concreti ciò che ci si aspetta.

Il tipo di criterio utilizzato corrisponde al livello di granularità del requisito:

- A livelli di astrazione più elevati (visione, obiettivi e epic), sono solitamente definiti i Criteri di Successo [SAFe], perchè è possibile misurare solo se la funzionalità/capacità richiesta viene fornita o meno.
- A livelli di astrazione più bassi, i Criteri di Accettazione possono essere utilizzati per descrivere come la soluzione verrà testata rispetto al requisito per ottenerne l'accettazione.

I metodi non-Agile e Agile concordano sul fatto che i requisiti devono essere verificabili. I metodi non-Agile usano spesso termini come "criteri di qualità" o "criteri di adeguatezza" o semplici "casi di test"; in Agile i termini "Criteri di Accettazione" (per le user story) o "Criteri di Successo" (per epic o temi) sono più comuni.

3.1.10 Definizioni di Ready e Don

Mentre i Criteri di Accettazione e di Adeguatezza appartengono e completano i requisiti di business, i prodotti del lavoro, "Definizione di Ready" (DoR) e "Definizione di Done" (DoD), supportano il processo formale di sviluppo e garantiscono la qualità dei requisiti e degli incrementi di prodotto. Il DoD è una parte ufficiale della Guida Scrum [Scrum2020] e viene descritto come l'impegno per l'incremento. Funziona come un filtro di qualità per il processo di sviluppo Agile, mentre la DoR è una buona pratica che dovrebbe aiutare a creare requisiti di valore ed evitare di sovraccaricare il team con requisiti non qualificati. La DoR aiuta a trattare i requisiti al giusto livello di dettaglio e a fornire informazioni sufficienti per la negoziazione tra il Product Owner/Requirements Engineer e gli sviluppatori.

3.1.11 Prototipo vs. Incrementi

Un altra modalità di trattare i requisiti è la creazione di prototipi, poiché molti stakeholder che hanno requisiti per un sistema o un prodotto non vogliono scrivere o leggere documenti – vogliono un risultato immediato. Per queste persone, il modo migliore per capire le loro esigenze e ottenere feedback è la dimostrazione delle funzionalità o capacità del sistema mediante un sistema funzionante.

Un modo per farlo è attraverso l'uso di prodotti minimali e incrementali. Il RE propone due tipi: incrementi di prodotto "orizzontali", per mostrare più elementi per la validazione ("fare più cose giuste") e incrementi di prodotto "verticali", per verificare che l'approccio di sviluppo sia corretto ("fare la cosa giusta").

Fornendo un'interazione diretta con un sistema funzionante, i prototipi possono essere molto utili per ottenere feedback. Le proprietà di usabilità, come ad esempio il tempo di reazione, sono difficili da specificare, ma facili da identificare quando si utilizza un software funzionante. I prototipi possono anche essere una fonte di frustrazione – per gli utenti perché credono che lo sviluppo sia già finito e per gli sviluppatori perché sono spesso allontanati per essere poi sostituiti con una tecnologia migliore.

I metodi Agile cercano di evitare i prototipi "usa e getta", sviluppando fin da subito incrementi di buona qualità del sistema o del prodotto "reale". Agile persegue iterazioni brevi, offrendo incrementi di prodotto dimostrabili, che vengono a loro volta utilizzati per ottenere ulteriori informazioni sui requisiti. Sebbene l'intenzione non sia quella di gettare via il codice sviluppato, il refactoring è una buona pratica Agile, come risposta al cambiamento dei requisiti funzionali o di qualità in base al feedback degli utenti.

Il termine "spike" in Agile viene utilizzato per riferirsi a un'iterazione di sviluppo eseguita con lo scopo esplicito di capire meglio un'area complessa (ad esempio l'architettura del sistema) e quindi ridurre il rischio. Il termine, sebbene non definito con precisione, può riferirsi alla validazione di un task o di un'intera iterazione. La prototipazione è una tecnica valida all'interno di uno spike, dove a differenza di altre iterazioni Agile l'obiettivo principale è la conoscenza acquisita piuttosto che il codice funzionante.

3.1.12 Sintesi dei prodotti di lavoro

Come menzionato negli ultimi paragrafi, alcuni prodotti del lavoro sono molto importanti per uno sviluppo di successo:

- È una buona pratica iniziare con visioni o obiettivi.
- È una buona pratica identificare e conoscere gli stakeholder più importanti.
- È buona pratica impostare esplicitamente l'ambito e delimitarlo nel contesto.

Anche se RE nei contesti non-Agile e Agile può utilizzare termini diversi, in entrambi i casi si concorda sul fatto che è necessario comprendere sia la funzionalità che i termini di business per acquisire i requisiti funzionali (comprese funzionalità e dati).

Inoltre i metodi Agile e non-Agile dovrebbero identificare requisiti di qualità e vincoli, in quanto possono influenzare fortemente le decisioni di progettazione.

Ignorarli o riconoscerli troppo tardi può portare a molte rielaborazioni e a un prodotto che non soddisfa le aspettative dei clienti.

Un buon RE assicura che non vengano dimenticati problemi rilevanti. I metodi non-Agile perciò insistono spesso nel documentare le diverse potenziali aree di interesse relative ai requisiti di un sistema.

Gli approcci Agile utilizzano meno prodotti del lavoro formali e sostituiscono la documentazione mancante con la comunicazione diretta e cicli di feedback veloci resi possibili tramite lo sviluppo di prodotti incrementali o prototipi.

Il secondo principio spesso citato del manifesto Agile [AgileMan2001] spiega esattamente quanto abbiamo illustrato:

„... Grazie a questo lavoro siamo arrivati a considerare importanti: ... Il software funzionante più che la documentazione esaustiva ..."

Siamo convinti che una valutazione deliberata di ciò che deve essere catturato per iscritto, di ciò che può essere discusso e di ciò che può essere prototipato o mostrato in incrementi è molto utile per qualsiasi organizzazione. I migliori risultati saranno raggiunti quando documentazione, comunicazione e prototipazione o sviluppo incrementale saranno bilanciati in base ai vincoli e alla cultura dell'azienda. Il capitolo 4 tratterà più a fondo il tema del bilanciamento delle attività iniziali (e dell'architettura) grazie al ricorso ad attività iterative.

3.2 Tecniche in RE@Agile (L2)

In 3.1 avete imparato a conoscere importanti prodotti di lavoro sui requisiti. In questa unità educativa, imparerete le attività chiave da svolgere nel RE. Il manuale del CPRE Foundation Level [CPREFL2022] struttura queste attività nel modo seguente:

- Elicitazione dei requisiti
- Documentazione dei requisiti
- Validazione e negoziazione dei requisiti
- Requirements management (Gestione dei requisiti)

Le seguenti sottosezioni trattano queste attività da una prospettiva Agile.

3.2.1 Elicitazione dei requisiti

Il Requirements Engineering nello sviluppo Agile si basa su un'intensa comunicazione tra tutti gli stakeholder (inclusi gli sviluppatori) per elicitare i requisiti. Una comunicazione informale e diretta tra i membri del team può essere ottenuta con una buona combinazione di interviste e brainstorming. Le tecniche come "l'on-site customer" di XP possono avere altrettanto successo sotto forma di sessioni tra il Product Owner e gli stakeholder (in genere gli sviluppatori). L'obiettivo in tutti i casi è quello di ottenere più informazioni possibile su ciò che è veramente necessario.

Il RE offre una gamma di tecniche per individuare ed elicitare i requisiti molto più ampia di quelle che sono normalmente discusse nel contesto dello sviluppo Agile. Esse includono tecniche di Q/A (non solo interviste, ma anche questionari), tecniche di osservazione, tecniche basate su prodotti del lavoro (riuso, archeologia di sistemi, ecc.) [CPREFL2022] e tecniche di creatività come il brainstorming o il design thinking. Queste tecniche supportano l'idea di requisiti in formato User Story, che sono la base per una discussione strutturata piuttosto che una prescrizione per l'implementazione.

Come menzionato nella sezione 3.1.11, i prototipi e gli incrementi di prodotto sono un altro modo per saperne di più sui requisiti. Non appena viene presentato un incremento del prodotto nella revisione dello sprint o nel meeting dimostrativo, possono emergere nuove idee che sono inseribili direttamente nel Product Backlog e prioritizzate dal Product Owner.

Il Requirements Engineering nello sviluppo Agile può trarre grande beneficio dal Requirements Engineering applicato nei progetti e nello sviluppo di prodotti non-Agile, studiando le diverse tecniche di elicitazione e prendendo una decisione deliberata sulla combinazione di tecniche da applicare. Nonostante le comunicazioni intensive tra gli stakeholder e il feedback rapido attraverso gli incrementi di prodotto siano idee eccellenti, c'è molto di più per l'elicitazione. Se ci sono ad. es. centinaia o migliaia di stakeholder, la comunicazione verbale da sola non è sufficiente. Quando si cerca di scoprire requisiti innovativi, o requisiti di cui le parti interessate non conoscono nemmeno la possibilità, possono essere necessarie tecniche di creatività.

Quando si lavora con vincoli temporali ristretti (cioè senza avere abbastanza tempo per discussioni intensive), tecniche come le snow card potrebbero essere più efficaci. Quando si esplorano nuove tecnologie, gli spike (timeboxed, incrementi semplici per esplorare potenziali soluzioni) sono una buona idea.

3.2.2 Documentazione dei requisiti

In Agile i requisiti sono organizzati in un backlog. I requisiti possono essere documentati sotto forma di story card annotate con valore di business e priorità. I requisiti possono essere organizzati all'interno di story map o scomposti in storie più semplici. Il principio dietro le card è che la dimensione della card limita ciò che si può scrivere e aiuta a focalizzarsi sui dettagli principali.

Tuttavia la documentazione dei requisiti è considerata ancora un'attività importante per favorire la comunicazione tra tutti gli stakeholder. Il formalismo per la documentazione può essere minimizzato se i dettagli sono comunicati verbalmente.

La definizione di un livello di documentazione adeguato dipende da molti fattori, quali la dimensione dei progetti o prodotti, il numero di stakeholder coinvolti, i vincoli legali o le criticità sulla sicurezza fisica legate al progetto. Basandosi su tali fattori, i processi di sviluppo Agile cercano di evitare la documentazione in eccesso e di trovare un insieme minimo di documentazione di contenuto utile.

Mentre il lavorare con un Product Backlog "in evoluzione" è un modo efficiente per gestire la documentazione, esso non è sempre sufficiente. Per cui si introducono brevemente altri tipi di documentazione.

In una prospettiva RE, distinguiamo quattro tipi di documentazione:

1. **Documentazione per scopi legali:** alcuni domini o contesti di progetto (ad es. software nel settore sanitario o avionico) richiedono la documentazione di alcune informazioni (ad es. requisiti e casi di test) per determinati destinatari allo scopo di ottenere l'approvazione legale.

> Il principio per la creazione di documentazione per scopi legali è il seguente: la documentazione legalmente necessaria deve essere derivata dalle leggi o dagli standard applicabili ed è una parte inseparabile del prodotto.

2. **Documentazione ai fini di preservare le informazioni:** Alcune informazioni su un sistema hanno un valore duraturo che va oltre la fase iniziale di sviluppo. Esempi di informazioni di questo tipo sono gli obiettivi per cui è stato realizzato il sistema, i casi d'uso principali implementati, le decisioni prese durante lo sviluppo, ad es. per escludere determinate funzionalità. La documentazione ai fini di preservare le informazioni può diventare l'archivio condiviso del team, del prodotto o dell'organizzazione. Può ridurre la dipendenza dalla capacità di memoria dei singoli membri del team e può facilitare le discussioni sulle decisioni assunte in precedenza (ad es. "Perché abbiamo deciso di non implementarlo?").

> Il principio è il seguente: il team decide cosa documentare allo scopo di preservare le informazioni rilevanti.

3. **Documentazione per scopi di comunicazione:** una comunicazione efficace ed efficiente è uno strumento importante nei metodi Agile a causa della loro interattività e dei cicli di feedback brevi. Ci sono diverse situazioni nei progetti reali che possono ostacolare la comunicazione verbale diretta: team distribuiti, barriere linguistiche o restrizioni temporali per coloro che sono coinvolti. Inoltre le informazioni sono a volte così complesse che la comunicazione diretta può essere inefficiente o fuorviante. Un prototipo cartaceo o un diagramma di un algoritmo complesso possono ad esempio essere riletti anche in un secondo momento. A volte gli stakeholder preferiscono semplicemente la comunicazione scritta alla lettura del codice sorgente o alla revisione del software. In questi casi la documentazione facilita il processo di comunicazione tra tutte le parti coinvolte e conserva i risultati.

Il principio per la creazione di documentazione a fini di comunicazione è il seguente: un documento viene creato come ulteriore mezzo di comunicazione se gli stakeholder o gli sviluppatori riconoscono un valore nell'esistenza della documentazione. Il documento deve essere archiviato quando la comunicazione è andata a buon fine.

4. **Documentazione per scopi di ragionamento:** un aspetto spesso dimenticato nello scrivere un documento è che la scrittura è sempre un mezzo per migliorare e supportare i processi mentali di chi scrive. Anche se il documento verrà cestinato più avanti nel processo, il beneficio di migliorare e supportare il pensiero è duraturo. Per es. scrivere uno use case costringe chi scrive a pensare alle interazioni concrete tra il sistema e gli attori, includendo ad esempio eccezioni e scenari alternativi. La scrittura di uno use case può quindi essere intesa come uno strumento per testare la propria conoscenza e comprensione di un sistema.

Il principio per creare documentazione a fini di ragionamento è il seguente: il pensatore decide sulla forma del documento che meglio supporta il suo modo di pensare. Il pensatore non ha bisogno di giustificare la forma di documentazione scelta per il ragionamento. Il documento può essere cestinato al termine del processo di ragionamento.

I metodi Agile possono trarre notevoli vantaggi se questi quattro tipi di documentazione sono identificati e applicati nel contesto appropriato. Riassumendo, possiamo dire che la documentazione dei requisiti non è fine a sé, ma dovrebbe facilitare la comunicazione tra gli stakeholder, in particolare tra il richiedente (spesso sostituito dal Product Owner) e gli sviluppatori.

3.2.3 Validazione e negoziazione dei requisiti

Mentre il RE enfatizza la validazione dei requisiti tramite metodi come revisioni, walkthrough, ispezioni o letture strutturate, i metodi Agile si sforzano di validare i requisiti attraverso feedback precoci e frequenti su incrementi di prodotti di valore. Una buona pratica per supportare questa modalità di validazione è l'uso di un test di regressione automatizzato, che fornisce una validazione continua dello sviluppo e dei relativi requisiti. L'obiettivo della validazione dei requisiti comprende l'identificazione dei requisiti mancanti, ambigui o errati,

nonché dei requisiti controversi o conflittuali per i quali è possibile applicare tecniche di negoziazione e risoluzione dei conflitti.

Dato che lo sviluppo incrementale e iterativo è una strategia chiave nei metodi Agile, la necessità di una validazione formale dei documenti diminuisce. Quest'ultima viene sostituita da una negoziazione costante sui requisiti tra tutti gli stakeholder, in modo che i conflitti vengano individuati e risolti fin dall'inizio.

La validazione formale viene ridotta anche mostrando risultati rapidi sotto forma di incrementi di prodotto integrati. Se l'incremento (delta) non soddisfa tutti i requisiti di tutti gli stakeholder, il delta viene reinserito nel Product Backlog sotto forma di nuovi requisiti e valutato e classificato con priorità rispetto a tutti gli altri elementi del backlog.

Tuttavia, la consultazione del Product Backlog, le discussioni sul valore di business, le discussioni sui rischi e le negoziazioni immediate sui requisiti sono tutte tecniche preziose nel RE@Agile. Tutte queste tecniche possono essere utilizzate all'interno di riunioni di affinamento, in cui il Product Owner e gli sviluppatori (e gli stakeholder se disponibili) lavorano insieme per trovare il livello di dettaglio necessario per l'implementazione, utilizzando i principi dell'affinamento continuo.

Nelle prime versioni della Guida Scrum, la cosiddetta riunione di affinamento del Product Backlog è stata menzionata solo indirettamente. Nella versione attuale [Scrum2020], il perfezionamento è esplicitamente menzionato come attività per l'elicitazione, la documentazione e la validazione dei requisiti. Questo processo di affinamento, di cui la riunione di affinamento è una parte essenziale, permette di scoprire i potenziali problemi in anticipo e di ridurre il tempo necessario per la riunione di pianificazione dello Sprint.

3.2.4 Requirements management (gestione dei requisiti)

Nel RE tradizionale la gestione dei requisiti si occupa di tutte le attività per gestire nel tempo i requisiti. Essa include la gestione delle versioni, la gestione delle modifiche, la gestione della configurazione, la tracciabilità, l'aggiunta di attributi, come stato, stime, priorità, collegamenti a requisiti in conflitto, nonché le persone coinvolte nell'acquisizione, controllo, approvazione, implementazione o testing dei requisiti.

Come discusso in precedenza (vedi 3.1.1), i prodotti di lavoro chiave per il mantenimento dei requisiti nei processi di sviluppo Agile sono raccolti in un backlog. A differenza della tradizionale gestione dei requisiti, i backlog sono progettati per mantenere solo la versione più recente e migliore di tutti i requisiti ancora da implementare. Gli elementi di backlog vengono in genere eliminati o archiviati non appena viene consegnato il prodotto che soddisfa tali requisiti.

Le attività di gestione dei requisiti che si svolgono nel backlog includono:

1. Prioritizzazione dei requisiti: determinare il loro valore (di business) per decidere quando implementarli. Più alto è il valore di business, più alta è la priorità del requisito, poiché i progetti Agile cercano di consegnare per primi gli elementi con il più alto valore di business. I fattori di influenza che determinano il valore del business e la

definizione delle priorità che ne derivano sono trattati nel CPRE Modul RE@Agile Practitioner e Specialista [CPREALAGILE2022].

2. Stima dei requisiti: determinare quanto lavoro è necessario per soddisfarli. Stime troppo elevate sono un chiaro messaggio a un Product Owner per comunicare che è necessario più lavoro per portarli alla definizione di ready (DoR, vedi 2). Quindi deve dividere il rispettivo item in unità più piccole per consentire una stima.

Questo non vuol dire che altre attività relative agli aspetti temporali della gestione dei requisiti non possano aver luogo in Agile. Tuttavia tali attività di solito vengono registrate al di fuori del backlog.

Nel decidere quali attività di gestione dei requisiti siano appropriate in un determinato contesto, il Requirements Engineer deve cercare un equilibrio tra la riduzione al minimo del sovraccarico, permettendo rilasci anticipati di soluzioni operative, e le esigenze a lungo termine dell'organizzazione, quali conformità normativa, documentazione operativa o passaggio di consegne a nuovi membri del team.

Conclusioni

Le attività sui requisiti quali elicitazione, documentazione, validazione e negoziazione, così come la gestione dei requisiti, devono essere eseguite anche nello sviluppo Agile. Le tecniche preferite per l'elicitazione e la documentazione possono differire tra lo sviluppo Agile e quello non-Agile, ma l'apprendimento reciproco permette di trovare la soluzione migliore, in quanto ne combina la forza riducendo gli sprechi o l'impegno inutile. Questo modo di lavorare riflette i cinque valori di Scrum (impegno, concentrazione, apertura, rispetto e coraggio) e la loro rappresentazione nei tre pilastri di Scrum (trasparenza, ispezione e adattamento) [Scrum2020].

Soprattutto l'apertura e il rispetto sono utili per portare l'esperienza di RE nel mondo di Agile e portare le idee dei principi di Agile nel patrimonio di RE.

In questa unità didattica, avete imparato che anche nel RE@Agile ci sono diversi prodotti del lavoro in più oltre alle user story nel Product Backlog e che le attività chiave di Requirements Engineering non devono essere dimenticate – ma possono essere eseguite con enfasi e metodi diversi – basati sui principi Agile spiegati in 1.

4 Aspetti organizzativi di RE@Agile (L2)

Durata: 1 ½ ora

Obiettivi Formativi

EO 4.1.1 Comprensione dell'interazione tra la struttura organizzativa e RE@Agile
EO 4.2.1 Conoscenza dell'interazione con gli stakeholder nei processi di RE Agile
EO 4.2.2 Conoscenza di come la Comunicazione e la Collaborazione possono aiutare a migliorare i risultati
EO 4.2.3 Conoscenza del Ruolo del Managment in Agile
EO 4.3.1 Conoscenza della motivazione per lo scaling
EO 4.3.2 Conoscenza delle dimensioni per l'organizzazione dei team
EO 4.3.3 Conoscenza degli approcci per organizzare la comunicazione tra i team
EO 4.3.4 Conoscenza degli esempi di framework per lo scaling
EO 4.3.5 Conoscere degli impatti principali dello scaling in RE
EO 4.4.1 Conoscenza dei criteri per decidere il livello di anticipo rispetto a un RE continuo
EO 4.4.2 Conoscenza del giusto livello di dettaglio per gli elementi del backlog
EO 4.4.3 Conoscenza del valore della validazione in RE@Agile
EO 4.4.4 Conoscenza dei corretti cicli di aggiornamento per il Product Backlog
EO 4.4.5 Conoscenza di come trovare la giusta tempistica per il ciclo di sviluppo

4.1 Influenza dell'organizzazione in RE@Agile (L2)

Agile ha le sue radici nel processi di controllo manifatturieri e empirici (vedi [TaNo1986]). I principi Agile sono facili da comprendere; le pratiche e i framework Agile, come Scrum ad esempio, sono facili da utilizzare in ambienti nuovi come le startup o le piccole aziende. È difficile però metterli in pratica in organizzazioni più grandi, che si comportano come un organismo vivente, che si difende da ogni intruso. D'altra parte, se le organizzazioni scoprono i benefici dell'applicazione di tali principi e pratiche, esse cercano poi di mescolarli con il loro DNA, come descritto nella legge di Conway [Conw1968]. Ciò porta a un crescente interesse per argomenti come la gestione Agile (vedi [ACP/PMI]) e le organizzazioni Agile (vedi [Denn2015], [Appel2011]), che portano a discussioni su Agile che spesso si estendono oltre lo sviluppo (software).

Idee quali mettere il cliente al centro, i gruppi auto-organizzati, l'abilitazione e la responsabilizzazione degli individui e il miglioramento continuo hanno una risonanza nel più ampio mondo aziendale. Il framework Scrum at Scale è un'estensione minima dello schema base Scrum, che mantiene la struttura modulare al centro dello schema Scrum e consente lo scaling di un'implementazione Scrum commisurata alle esigenze specifiche dell'azienda.

Nel mondo dello sviluppo software, molte implementazioni dei processi di sviluppo Agile falliscono perché il resto dell'organizzazione non è stato in grado di cambiare per supportare i team Agile.

Perché è necessario un tale cambiamento organizzativo?

Riportiamo due esempi del perché anche il resto dell'azienda deve cambiare per supportare uno sviluppo Agile:

1. L'organizzazione (o sue componenti) nella domanda di nuovi sviluppi deve essere in grado di fornire requisiti sufficientemente buoni (buoni significa abbastanza dettagliati, senza esserlo eccessivamente – seguendo la Definizione di Ready) per mantenere lo sviluppo attivo a un ritmo costante. Questo, in combinazione con i requisiti che cambiano di frequente, richiede un flusso di domanda continuo.
2. Il dipartimento risorse umane deve capire quali persone assumere per supportare correttamente i team Agile. Spesso si trovano offerte di lavoro per Scrum Master fuorvianti, in cui sono richieste competenze e certificazioni di programmazione, il che dimostra che i principi dei tre ruoli di Scrum sono fraintesi.

In 4.2 vengono esaminati gli aspetti organizzativi di quando si incorpora un'unità organizzativa Agile, come un team Scrum, in un ambiente non-Agile. In molti casi, sebbene l'introduzione di Agilità inizi nello sviluppo del prodotto, l'intera azienda potrebbe non seguire necessariamente i principi Agile.

L'influenza sull'organizzazione nel caso in cui sia necessario più di un team Agile per risolvere un problema complesso è discussa in 4.3. L'elemento centrale è di nuovo l'organizzazione IT e le sue interfacce per il business. La transizione a un'organizzazione completamente Agile non è qui considerata esplicitamente in quanto non rientra nell'ambito di una discussione su RE.

4.4 si concentra sugli aspetti organizzativi temporali, analizzando in particolare la questione di quando dovrebbero essere eseguite le attività di RE.

4.2 Sviluppo Agile in un ambiente non-Agile (L1)

4.2.1 Interazione con gli stakeholder esterni all'organizzazione IT

Il ruolo dell'unità di sviluppo all'interno dell'azienda è quello di fornire soluzioni e servizi ai clienti aziendali (sia all'interno che all'esterno dell'organizzazione).

Agile pone il cliente al centro dello sviluppo del prodotto. Questo significa che il cliente è coinvolto nell'intero ciclo di vita dello sviluppo del prodotto, che il feedback sulle consegne incrementali viene attivamente ricercato e che nuovi requisiti in linea con le esigenze del business sono accettati.

Con il coinvolgimento continuo del cliente, anche RE diventa un processo continuo (vedi 2.4). La persona responsabile, come il Product Owner, dovrebbe coinvolgere i propri clienti in una comunicazione aperta e diretta, ascoltando le nuove esigenze e le mutevoli aspettative e acquisendole all'interno del backlog.

Nella pratica questa comunicazione può assumere molte forme. Se il cliente è effettivamente disponibile a lavorare con il team su base giornaliera, allora la comunicazione può essere davvero diretta e per lo più di natura informale, dove sono documentati solo risultati o decisioni.

È importante riconoscere la presenza di fattori al di fuori del controllo degli sviluppatori (ad es. la distanza geografica o semplicemente la mancanza di disponibilità), il che significa che l'interazione diretta non è sempre praticabile. In questi casi si deve pianificare una forma più efficiente di comunicazione, sia invitando i rappresentanti dei clienti a riunioni regolari di pianificazione e revisione, sia eseguendo sessioni intensive a tempo vincolato – timeboxed (ad es. design thinking o design Sprint, vedi 1.5) in una fase iniziale con l'input risultante catturato poi nel backlog.

4.2.2 Organizzazione per prodotto vs. per progetto

Agile promuove relazioni dirette, aperte e non gerarchiche all'interno dell'organizzazione e flessibilità per tener conto dell'evoluzione dei prodotti nel tempo. Le organizzazioni più grandi, tradizionalmente basate su strutture di gestione top-down, attribuiscono un valore elevato alla pianificazione e alla prevedibilità. Il progetto e la pianificazione delle risorse, ad esempio, sono realtà alle quali le aziende non possono facilmente rinunciare.

Lo sviluppo del software spesso è stato tradizionalmente basato sui progetti, il che significa che si svolge come una serie di attività che impegnano temporaneamente un'organizzazione per la produzione di prodotti, servizi o risultati unici (vedi [PMI]). Gruppi di progetti correlati con i loro obiettivi o obiettivi condivisi vengono in genere definiti programmi, mentre la pianificazione e il controllo dell'intera gamma di progetti e programmi all'interno di un'organizzazione viene definita la gestione del portafoglio.

Fedele alle sue radici nello sviluppo del prodotto, l'approccio Agile è più incentrato sui prodotti. Viene mantenuto un backlog di miglioramenti del prodotto, che sono poi implementati in un processo iterativo di miglioramento continuo. Agile da solo non definisce una data di fine come tale. Poiché a lungo termine ci sono miglioramenti da apportare o benefici da realizzare, il lavoro dovrebbe, in linea di principio, continuare se i benefici superano lo sforzo o i costi.

Mentre questi approcci non si escludono a vicenda (lo scopo di un progetto può essere quello di fornire un particolare prodotto, ma vengono anche stabiliti ulteriori progetti per successivi miglioramenti) le differenze di prospettiva e terminologia possono essere fonte di tensione e di incomprensione tra uno sviluppo del software Agile e le organizzazioni non-Agile.

Un approccio per risolvere tali tensioni è che mentre lo sviluppo del software si svolge in modo rigorosamente Agile, le funzioni di portfolio e di gestione programmatica forniscono un livello superiore di pianificazione e controllo che utilizza un approccio mutuato da entrambi i mondi (vedi anche 4.3). Un fattore chiave per rendere funzionante tale soluzione è la capacità di colmare le lacune concettuali tra l'adempimento pianificato degli obiettivi e delle caratteristiche di business a livello di portafoglio e di programma e una consegna iterativa e flessibile delle singole funzionalità del software.

RE fornisce i concetti e i metodi necessari per differenziare i requisiti a diversi livelli di astrazione per i requisiti di business a livello di portfolio e di programma e perciò derivare requisiti dettagliati adatti al backlog di sviluppo. L'approccio ai requisiti spazia da requisiti più dettagliati e precisi utilizzati come ordini puntuali per lo sviluppo a requisiti che vengono utilizzati come base comune per discussioni e convergenze tempestive, nonché dettagliati quanto necessario per il livello di pianificazione corrente.

4.2.3 Il ruolo del management nel contesto Agile

Discipline & team: Il personale all'interno dei dipartimenti IT è stato tradizionalmente organizzato per disciplina: sviluppatori, tester, requirements engineer, analisti di business, project manager, ecc. I team di progetto vengono costituiti da questi diversi insiemi di competenze per la durata fissata per il progetto. Agile, e Scrum in particolare, promuovono l'idea di più team funzionali trasversali. Al di fuori dei ruoli specialistici di Product Owner e Scrum Master, tutti i membri del team dovrebbero essere in grado di operare con diverse capacità, interagendo con le diverse figure professionali che supportano il RE o le attività di test necessarie. Anche per questo motivo, questi membri del team sono comunemente chiamati "sviluppatori" in Scrum. L'obiettivo del team è quello di essere in grado di soddisfare pienamente le esigenze dei clienti, indipendentemente dagli elementi tecnologici o organizzativi coinvolti. Questo risultato può essere ottenuto quando si hanno competenze RE tra gli sviluppatori (soluzione preferibile) o fuori dal team (spesso utilizzate in realtà come supporto al Product Owner), di risorse quindi che non fanno ufficialmente parte dello schema Scrum.

IT Manager: è compito dei manager IT (chiamati "people developer" in [SAFe]) trovare il giusto equilibrio nella loro organizzazione tra competenze specialistiche e generiche e fornire supporto per organizzare tali competenze in un numero appropriato di team. Per quanto riguarda la legge di Conway [Conw1968], la struttura del team rispecchierà la struttura del prodotto (componenti) o della struttura del sistema in IT. Questo aspetto è ancora più importante nel dimensionamento del numero di team. Se un'azienda organizza i propri team per componenti o sistemi, uno scaling, attuato con l'aumento del numero di team, non aiuta poiché aumenta le dipendenze. Lo scaling sarebbe possibile solamente aumentando il numero di membri del team, ma solo fino al punto in cui lo sforzo di comunicazione diventerebbe insostenibile.

Team inter-funzionali, che includano tutte le competenze per fornire interi incrementi dal frontend al backend, possono essere facilmente dimensionati – ma non sono facili da costituire e per farlo potrebbe essere necessario molto tempo.

Product Owner vs. Project Manager: Come discusso in precedenza, il ruolo di Product Owner amplia le responsabilità dei Requirements Engineer, dato che deve assumersi anche la responsabilità di definire le priorità di business nei requisiti forniti ai team di sviluppo del prodotto. I Product Owner devono essere in grado (avendo le opportune conoscenze) ed essere autorizzati a prendere decisioni aziendali. Alcune decisioni, precedentemente prese dai project manager, non sono quindi più necessarie quando si lavora con approcci Agile, poiché gli sviluppatori si auto-organizzano nei loro confini organizzativi e hanno solo bisogno

di sapere quale sia il lavoro da svolgere (requisiti a un livello di dettaglio che consenta lo sviluppo all'interno di una iterazione) per organizzare il proprio lavoro autonomamente (suddivisione delle attività e assegnazione all'interno del team).

Quello che è richiesto ai project manager IT Agile è una chiara impostazione della visione per lo sviluppo Agile e una chiara comunicazione dei presupposti culturali per il successo di questo approccio.

Trovare il giusto equilibrio e soddisfare le aspettative dell'azienda non è semplice. Alcuni criteri di successo sono discussi di seguito in 4.4.

Rilevanza dell'Ingegneria dei requisiti (RE): I modelli precedenti si applicano anche a Scrum come esempio. Le persone che lavorano secondo i principi di RE possono far parte del team e quindi non saranno nominate separatamente. In alternativa, possono formare loro stessi un team che supporta uno o più Product Owner come squadra. Entrambi gli approcci hanno i loro vantaggi e possono essere usati in combinazione senza violare i principi Agile. Il Requirements Engineer può così diventare la spina dorsale di uno sviluppo Agile di successo.

4.3 Gestione di problemi complessi mediante scaling (L1)

4.3.1 Motivazione per lo scaling

I valori Agile della comunicazione diretta, giornaliera e non-gerarchica sono in genere presenti in team piccoli e affiatati, come i team Scrum con un numero raccomandato di non più di 10 membri (developers + Product Owner + Scrum Master). I membri del team dovrebbero idealmente essere fisicamente nello stesso luogo e inter-funzionali in termini di capacità tecniche e di business. Nelle organizzazioni più grandi questa visione ideale dello sviluppo Agile potrebbe non essere fattibile per molte ragioni:

- Problemi complessi possono coinvolgere gli stakeholder e le conoscenze provenienti da diverse parti del business e che non possono essere facilmente raccolte in un singolo team.
- Problemi complessi possono coinvolgere una serie di tecnici specializzati, le cui conoscenze possono non essere facilmente raccolte/disponibili in un singolo team.
- L'ambito richiesto dalla data di rilascio fissata può essere al di là della velocity con cui può lavorare un singolo team.
- Il personale che opera all'interno di aziende globali può essere distribuito geograficamente.

Il termine Scaled Agile viene utilizzato per descrivere situazioni in cui più team sono tenuti a collaborare su un unico prodotto o soluzione condividendo obiettivi comuni. Gli approcci di Scaled Agile richiedono decisioni su come organizzare i team e su come coordinare le comunicazioni tra i team. L'obiettivo è quello di ottenere un approccio efficace per la gestione di problemi complessi, pur mantenendo il maggior numero possibile di vantaggi dell'Agilità.

4.3.2 Approcci per l'organizzazione di team

La domanda a cui rispondere è come un'organizzazione dovrebbe strutturarsi in team di una dimensione che permetta ai vari team di essere inter-funzionali (dimensione minima), ma allo stesso tempo efficaci (dimensione massima) per quanto riguarda la comunicazione e l'allineamento. L'organizzarsi lungo le linee funzionali (ad es. un team specializzato in un'area di business definita, o anche su una funzionalità all'interno di un'area di business) ha il vantaggio di concentrare le conoscenze di business all'interno di un singolo team. Ciò potrebbe facilitare l'elicitazione dei requisiti, ad es. riducendo il numero di stakeholder di business direttamente coinvolti nel team.

Uno svantaggio di questo approccio è che l'offerta di funzionalità end-to-end in un'area di business può coinvolgere diverse specializzazioni tecniche, (come progettazione di interfacce utente, motori di processo, database e piattaforme core come ERP o mainframe) che possono facilmente superare la dimensione massima di un team efficace.

Un modo alternativo di organizzare lo sviluppo è lungo linee tecniche, con team specializzati in componenti tecniche o piattaforme. Il vantaggio dei team di componenti o piattaforme è quella di avere una conoscenza approfondita nel rispettivo campo tecnologico. Lo svantaggio è rappresentato dalle dipendenze che un simile approccio crea tra team che lavorano insieme per fornire l'intero incremento del prodotto da pianificare.

Gli schemi di scaling (come quelli indicati in 4.3.4) mostrano le modalità per gestire situazioni di questo tipo. Le attività di RE devono essere allineate al framework per ottenere una visione comune di ciò che può essere fornito.

In questo scenario, RE ha un ruolo particolarmente importante da svolgere, suddividendo prima gli obiettivi e i requisiti di business in requisiti dei sottosistemi costituenti, che possono poi essere assegnati ai singoli team, e in secondo luogo seguendo costantemente lo sviluppo per garantire il risultato con una soluzione integrata e l'effettivo ottenimento del valore di business.

Una combinazione di tutte le tipologie di team può fornire la soluzione migliore e più pragmatica, per trarre vantaggio da idee dei team funzionali, tenendo però anche conto dei vincoli specifici dell'azienda (per dettagli vedere [CPREALAGILE2022]).

4.3.3 Approcci per l'organizzazione della comunicazione

Per rispondere alla domanda su quanti team possono lavorare insieme in modo efficiente, possiamo distinguere tra due diversi approcci:

1. Utilizzo di metodi da approcci di team singoli
2. Introduzione di concetti aggiuntivi per organizzare le comunicazioni e definire le responsabilità

Utilizzo di metodi da approcci di team singoli: il primo approccio si basa sull'idea che non sono necessari ulteriori prodotti del lavoro e ruoli e che la comunicazione tra diversi team dovrebbe essere supportata con le tecniche esistenti per gli approcci di un singolo team. Ruoli e prodotti del lavoro aggiuntivi sarebbero contrari al pensiero Agile e porterebbero

ulteriore complessità nell'organizzazione. Non deve quindi essere creato un sovraccarico dovuto a nuovi ruoli, osservando il principio di base "mantieni la semplicità". La comunicazione e il coordinamento tra i team sono di solito avviati dai Product Owner dei team, ma realizzati dai delegati del team. Costrutti come Comunità di Pratiche consentono ai membri del team, per tutti i team, di condividere esperienze e di coordinare i processi complessivi.

Introdurre ulteriori concetti: Il secondo approccio raccomanda di dividere i problemi più grandi in problemi più piccoli e di gestire le responsabilità in base a ruoli diversi per astrazioni diverse. Questi ulteriori prodotti del lavoro dovrebbero essere introdotti per diversi livelli di astrazione (ad es. Epic di Business, Epic di Architettura, Temi di Investimento, Features, User Story).

A seconda del framework utilizzato, vengono introdotti ruoli aggiuntivi con responsabilità per i diversi livelli di astrazione (ad esempio, Portfolio Manager, Product Manager e Product Owner). A causa della crescente complessità, sono necessari anche ulteriori prodotti del lavoro e ruoli per gestire la pianificazione e la comunicazione tra i diversi team e per ottenere risultati integrati per ogni iterazione (ad es. Roadmap e Release Manager). Inoltre devono essere organizzati incontri specifici per promuovere la comunicazione tra i ruoli nuovi e quelli esistenti. RE fornisce molte tecniche che potrebbero aiutare a suddividere i problemi in sotto-problemi e a supportare i nuovi ruoli ai diversi livelli di astrazione (ad es. modellazione del contesto, modellazione degli obiettivi).

Entrambi gli approcci hanno i loro vantaggi e svantaggi e ogni utente/azienda ha bisogno di trovare la modalità migliore in base al business case.

4.3.4 Esempi di framework per lo scaling di RE@Agile

Framework che supportano lo scaling di Scrum e Agile: sono disponibili molti e diversi schemi (framework) per supportare gli approcci di scaling e, data la crescente importanza del dimensionamento dell'Agilità, il loro numero è anche in rapida crescita. Troverete una selezione degli schemi più noti di seguito:

- **Scaled Agile Framework (SAFe)**
  SAFe è una base di conoscenza di comprovati modelli di successo per la realizzazione di software snello e flessibile e per lo sviluppo di sistemi su scala aziendale [SAFe]
- **Large-Scale Scrum (LeSS)**
  LeSS è lo Scrum applicato a molti team che lavorano insieme su un unico prodotto. [Less]
- **Nexus**
  Nexus è uno schema che guida al cuore dello scaling: dipendenze inter-team e problemi di integrazione [Nexus2015]
- **Lo Scrum degli Scrum**
  Scrum of Scrums è una tecnica in cui Scrum viene utilizzato per coordinare più team. [SofS] Ogni squadra assegna una persona (un ambasciatore) che la rappresenti nelle riunioni di coordinamento, che normalmente si tengono due o tre volte a settimana [CPREALAGILE2022].

- **Scrum@Scale**
  Il framework Scrum at Scale è un'estensione minima dello schema base Scrum, che mantiene la struttura modulare al centro dello schema Scrum e consente lo scaling di un'implementazione Scrum commisurata alle esigenze specifiche dell'azienda [SatS].

4.3.5 Impatti dello scaling su RE@Agile

I livelli di astrazione discussi sopra indicano che le attività di RE sono gestite da più ruoli, come Chief Product Owner, Product Manager, Portfolio Manager, Business Analyst. Ogni ruolo creerà prodotti del lavoro RE corrispondenti alla rispettiva area di interesse (Epic, Feature, User Story e la tracciabilità tra di essi). Sono necessari riunioni aggiuntive per le attività di RE (ad es. story time, feature time, demo di sistema), mentre le comunicazioni con gli stakeholder sono eseguite da diversi ruoli e a vari livelli all'interno dell'organizzazione.

Poiché lo stesso Scrum non si ampia facilmente per conto suo, RE svolge un ruolo fondamentale nel determinare in che modo i requisiti generali vengono scomposti e distribuiti in modo appropriato tra più team Agile, al fine di mantenere valido qualsiasi approccio di dimensionamento. Le tecniche di RE aiutano a strutturare l'analisi dei problemi e ad affinare i requisiti ad alto livello in requisiti dettagliati, come feature e user story appropriate per i singoli team. Un approccio strutturato, applicando astrazioni appropriate e diverse prospettive di analisi, fornirà una base solida per una suddivisione delle attività tra i team Agile semi-autonomi.

Ciò si applica in particolare alle dipendenze all'interno dei requisiti, che devono essere trovate e discusse il prima possibile per evitare sprechi nello sviluppo.

Un'ulteriore sfida può verificarsi se i team o altri ruoli lavorano in luoghi diversi. La complessità della gestione della comunicazione aumenta non solo a causa di un possibile orario di lavoro basato su fusi diversi o dell'utilizzo di lingue diverse. Nella maggior parte dei casi, la sfida principale è legata alla presenza di culture diverse. Poiché la comunicazione è uno dei compiti principali dei Product Owner e dei Product Manager, ciò ha un'influenza significativa sulle competenze richieste dai ruoli rilevanti di RE.

4.4 Bilanciare RE anticipato e continuo nel contesto dello scaling (L1)

Ad un livello molto astratto, i metodi Agile possono essere descritti come un processo continuo e iterativo in cui il sistema viene sviluppato in modo incrementale in base agli elementi del backlog. Dal punto di vista di RE è possibile identificare cinque parametri (vedi i seguenti sottocapitoli) che guidano questo processo:

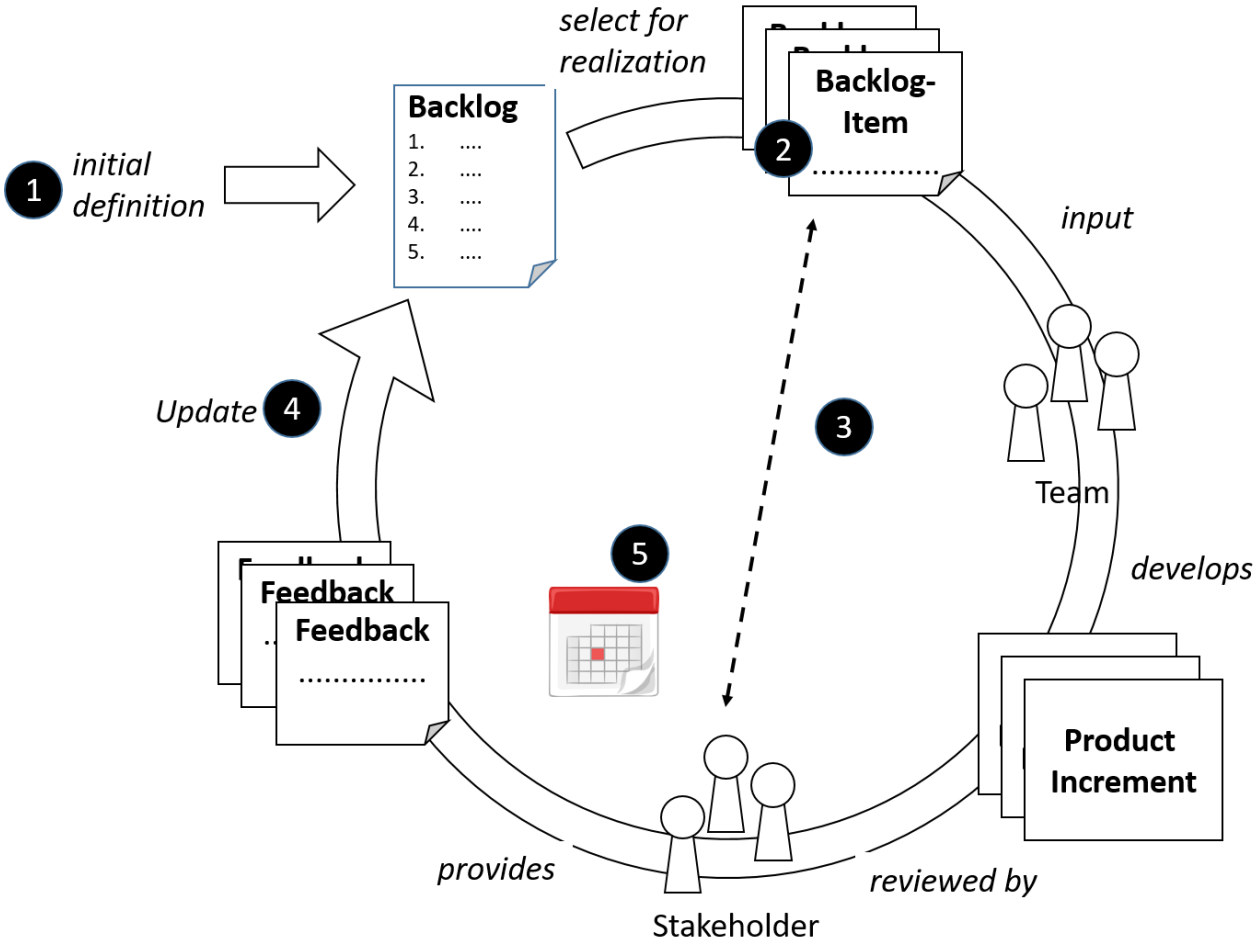

Figura 2: Processo continuo di RE

4.4.1 Definizione iniziale dei requisiti

Prima dell'avvio del processo di sviluppo continuo, è necessario creare un backlog iniziale (vedi [TaNo1986]). Questa definizione iniziale del backlog viene spesso definita "definizione anticipata". Il RE è talvolta interpretato come stabilire che questa definizione anticipata sia una specifica completa e dettagliata di tutti i requisiti. Questo può essere il caso di alcuni modelli di processo o domini, ma non è sempre così ed è piuttosto raro nello sviluppo Agile.

La base per la definizione iniziale del backlog è la quantità di informazioni necessarie per avviare la prima iterazione del processo di sviluppo continuo. Quando iniziare il processo di iterazione è quindi una decisione chiave. A seconda del sistema e del contesto, l'incertezza relativa ai requisiti (causata da una conoscenza non sufficiente) può comportare ulteriori ritardi, costi o potenzialmente un fallimento del progetto. Diversamente, l'iniziare troppo tardi potrebbe avere un impatto negativo sul time-to-market e sull'adeguatezza alle esigenze degli utenti finali in un determinato momento.

Il RE ci dice che dovrebbero essere elicitati e dettagliati per primi i requisiti che possono essere identificati come aventi un forte impatto sull'architettura, sulla fattibilità complessiva della soluzione o su scelte chiave riguardanti l'infrastruttura e l'hardware. Alcuni problemi rilevanti per l'architettura dovrebbero in effetti essere elicitati e analizzati prima della prima

iterazione dello sviluppo. I requisiti di minor impatto possono poi essere affinati durante le iterazioni del processo continuo.

Nell'ambito dei processi iterativi e incrementali di Agile, Il RE diventa un processo continuo che tempestivamente rilascia e perfeziona i requisiti, con dettagli sufficienti per alimentare il ciclo di sviluppo. Il processo assomiglia ad un imbuto per la frantumazione di pietre, dove grosse pietre entrano nell'imbuto, vengono ridotte a dimensioni più piccole e infine, al termine del processo, in piccoli sassolini.

4.4.2 Livello di dettaglio per gli elementi del backlog

Il livello di dettaglio di un backlog item limita la libertà degli sviluppatori nella realizzazione di tale elemento (vedi [Pich2010]). Tutti gli aspetti di un backlog item che non sono stati definiti vengono lasciati come decisioni per il team, il che dovrebbe consentire al team di essere più creativo, pur entro i limiti definiti dal business.

Le competenze disponibili degli sviluppatori possono servire come regola empirica. Se gli sviluppatori hanno competenze ed esperienze sufficienti per decidere i dettagli di un elemento del backlog (ad esempio, un esperto di autenticazione e autorizzazione è tra gli sviluppatori), la decisione sui dettagli dovrebbe essere lasciata agli sviluppatori. Si veda anche 3.1.5.

4.4.3 Validità dei Backlog Item

Conoscere la validità di un backlog item prima di implementarlo aiuta a minimizzare il lavoro di implementazione non necessario (vedi [Denn2015]). Rinviare il riconoscimento di un requisito errato o incompleto nel software sviluppato è un approccio molto costoso per la validazione dei requisiti.

La determinazione della validità di un backlog item è strettamente correlata al suo livello di dettaglio. La validità di un backlog item può essere determinata solo quando l'elemento stesso ha dettagli sufficienti. Pertanto, lo sforzo per elaborare e convalidare un requisito prima dell'implementazione deve essere confrontato con lo sforzo stimato per implementare il requisito e validarlo successivamente nel software rilasciato.

Quando la preferenza degli stakeholder relativa a un backlog item può essere determinata con uno sforzo accettabile, tale validazione dei requisiti dovrebbe aver luogo prima che venga sviluppato l'elemento in questione..

Esempi tipici di requisiti di questo tipo (incluso un approccio di validazione esemplificativo) sono: progettazione generale dell'interfaccia utente (ad es. con i mockup UI), meccanismi di autorizzazione e autenticazione (ad es. con revisioni dei casi d'uso), strutture di dati che devono essere memorizzate nel sistema (ad es. con revisioni di modelli di dati) e requisiti per interfacce verso sistemi esistenti (ad es. con revisioni degli activity diagrams).

Gli elementi di backlog ad alto rischio (ad es. funzionalità di business critiche, funzionalità critiche per la sicurezza fisica, funzionalità innovative) o con un costo dell'implementazione dei test elevato (ad es. il software deve essere testato in un prototipo costoso) devono essere validati prima della loro implementazione.

4.4.4 Feedback e Aggiornamento del backlog

I backlog vengono spesso aggiornati in base al feedback di un'attività di ispezione come ad esempio la sprint review in Scrum. Tale approccio è possibile per requisiti dettagliati o di piccola dimensione per i quali l'impatto di un cambiamento può essere analizzato e compreso rapidamente, in un ambiente con uno o due piccoli team. Non è consigliabile modificare i requisiti di maggiore complessità o che hanno dipendenze multiple con breve preavviso. In queste situazioni la modifica del backlog richiede più tempo, gli stakeholder potrebbero non essere disponibili e sono necessarie ulteriori analisi.

Un altro fattore che potrebbe avere un impatto sulla modifica degli elementi di backlog è il processo decisionale dell'organizzazione. In organizzazioni dove le decisioni significative possono richiedere del tempo (ad es. il consiglio responsabile si riunisce solo una volta ogni tre mesi), il principio dell'affinamento continuo deve assumere la forma di riunioni operative che coinvolgano tutti gli stakeholder. Queste riunioni hanno bisogno di RE nell'ambito della loro preparazione e come supporto alle decisioni da prendere.

4.4.5 Tempificazione del Ciclo di Sviluppo

Il parametro finale nel processo di sviluppo è la pianificazione temporale o la durata dell'iterazione. Tale pianificazione ha un effetto significativo sulle attività di RE che devono essere eseguite mentre si lavora su elementi di backlog non ancora in fase di sviluppo. Inoltre la durata dell'iterazione determina la frequenza con cui i risultati vengono consegnati agli stakeholder aziendali o ai clienti disponibili per la revisione.

Di conseguenza, le durate delle iterazione più brevi aumentano il carico di lavoro degli stakeholder aziendali per tre motivi (ad es .[Rein1997]):

1. Gli stakeholder aziendali devono essere disponibili per tutta l'iterazione quando si lavora sul backlog per creare input per l'iterazione successiva.
2. Gli stakeholder aziendali devono rivedere i risultati creati dal team al fine di fornire il loro feedback.
3. Gli stakeholder aziendali devono considerare un frequente cambio di contesto tra le loro attività quotidiane e il lavoro del progetto.

Iterazioni che richiedono tempi lunghi riducono la pressione, ma riducono anche la capacità di influenzare il backlog per lo sviluppo del prodotto.

La definizione della durata di una iterazione deve essere fatta tenendo conto della disponibilità degli stakeholder aziendali. Gli stakeholder aziendali in genere non sono disponibili al 100% per le attività di sviluppo, poiché hanno altri compiti all'interno dell'organizzazione per la quale è sviluppato il sistema.

Come regola generale, una durata di iterazione più breve fornisce feedback frequenti e maggiori opportunità di scoprire in anticipo gli errori, pertanto le iterazioni più brevi tendono ad accelerare le attività di sviluppo. Se un ciclo di breve durata rappresenta un carico inaccettabile per gli stakeholder, si dovrebbe trovare un compromesso sulla durata dell'iterazione, che tuttavia potrebbe accorciarsi in proporzione alla priorità del progetto.

Un altro fattore che può avere un impatto sul ciclo delle iterazioni è la dimensione media e la complessità degli elementi del backlog. Backlog item più grandi o più complessi richiedono più tempo per la comprensione e l'analisi. Pertanto il ciclo può essere allungato per gestire backlog item più grandi o più complessi all'interno di una singola iterazione. Ad esempio se il sistema in sviluppo è in una fase iniziale, un ciclo d'iterazione più lungo può essere consigliabile per dare al team più tempo per ottenere una comprensione iniziale del sistema. Tuttavia questo fattore deve essere bilanciato con l'obiettivo di utilizzare cicli più brevi per ottenere feedback frequenti. La decisione se avere cicli più o meno lunghi deve essere presa insieme da tutti i membri del team, soppesando le necessità dell'analisi con l'obiettivo di un feedback tempestivo. La modifica della durata di un ciclo si basa sempre sul principio "ispeziona e adatta", tenendo presente che l'esperienza passata non è sempre una garanzia per il futuro. Il cambiamento della durata di un'iterazione dovrebbe essere deciso prima del suo inizio, mai durante. Inoltre, il tempo di ciclo dovrebbe essere il più coerente possibile per ridurre la complessità e stabilire una capacità di previsione raggiungibile. Cambiare la durata troppo spesso può causare instabilità, diminuire l'impegno del team e molto probabilmente impedire un ritmo di incremento del prodotto fattibile.

5 Definizioni di Termini, Glossario (L2)

Il glossario definisce i termini rilevanti nel contesto di RE@Agile Primer. Il glossario è disponibile per il download nella homepage IREB all'indirizzo https://www.ireb.org/en/downloads/#re-agile-glossary

6 Riferimenti Bibliografici

[ACP/PMI] Agile Certified Practitioner: http://www.pmi.org/certification/Agile-management-acp.aspx. Ultima visita Giugno 2024.

[AgileMan2001] Agile Manifesto: http://www.Agilemanifesto.org, 2001. Ultima visita Giugno 2024.

[AmLi2012] Ambler S.; Lines, M.: Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise. IBM Press, 2012

[Ande2012] Anderson, D.J.: Lessons in Agile Management: On the Road to Kanban. Blue Hole Press, 2012

[Appel2011] Appelo J.: Gestione 3.0. Addison-Wesley Professional, 2011

[Beck2003] Beck, K.: Test-Driven Development by Example. Addison Wesley – Vaseem, 2003

[Beck2004] Beck, K.: Extreme Programming Explained: Embrace Change. Addison-Wesley Professional, 2004

[CNYM2000] Chung, L.; Nixon, B. A.; Yu, E.; Mylopoulos, J. : Non-Functional Requirements in Software Engineering. Springer Science & Business Media, 2000

[Cock1998] Cockburn, A.: Surviving Object-Oriented Projects. Addison-Wesley, 1998

[Cohn2004] Cohn, M.: User Stories Applied: For Agile Software Development. Addison Wesley Professional, 2004

[Conw1968] Conway, M.: How Do Committees Invent? Datamation 14(4):28–31, 1968. Article available at .http://www.melconway.com/Home/Conways_Law.html Ultima visita Giugno 2024.

[CPREFL2022] IREB e.V.: Syllabus CPRE Foundation Level, version 3.1. https://www.ireb.org/downloads/#cpre-foundation-level-syllabus-3-0. Ultima visita Giugno 2024.

[CPREALAGILE2022] IREB e.V.: Syllabus CPRE RE@Agile Practitioner | Specialist, versione 2.2.0. https://www.ireb.org/downloads/#cpre-advanced-level-re-agile-syllabus. Ultima visita Giugno 2024.

[Denn2015] Denning, S.: How To Make The Whole Organization Agile. http://www.forbes.com/sites/stevedenning/2015/07/22/how-to-make-the-whole-organization-agile/#658d3f65135b, 2015. Ultima visita Giugno 2024.

[Dsch2015] d.school: An Introduction to Design Thinking – Process Guide. https://web.stanford.edu/~mshanks/MichaelShanks/files/509554.pdf, 2015. Ultima visita Giugno 2024.

[Glinz2022] Glinz, M.: A Glossary of Requirements Engineering Terminology, https://www.ireb.org/downloads/#cpre-glossary. Ultima visita Giugno 2024.

[Griff2015] Griffiths, M.: PMI-ACP Exam Prep. Rmc Publications, 2015

[GoAk2003] Gordijn, J.; Akkermans, J.M.: Value-based Requirements Engineering: exploring innovative e-commerce ideas. Springer, 2003

[High2009] Highsmith, J.: Agile Project Management: Creating Innovative Products. Addison-Wesley Professional, 2009

[ISO25010] ISO/IEC Systems and software engineering – Systems and software Quality Requirements and Evaluation. ISO/IEC Standard 25010:2011

[KnZK2016] Knapp, J.; Zeratsky, J; Kowitz, B.: Sprint: How to Solve Big Problems and Test New Ideas in Just Five Days. Simon & Schuster, 2016

[Kron2008] Kronfaelt, R.: Ready-ready: the Definition of Ready for User Stories going into Sprint planning. http://scrumftw.blogspot.de/2008/10/ready-ready-definition-of-ready-for.html, 2008. Ultima visita Giugno 2024.

[Less] Large Scale Scrum: http://less.works. https://less.works/resources/LeSS-brochure.pdf Ultima visita 2024.

[LiOg2011] Liedtka, J.; Ogilvie, T.: Designing for Growth: A Design Thinking Tool Kit For Managers. Columbia Business School Publishing, 2011

[Martin1991] Martin, J.: Rapid Application Development. Macmillan Coll Div, 1991

[Meyer2014] Meyer, B.: Agile! – the good, the hype, and the ugly. Springer, 2014

[MeMi2015] Mesaglio, M., Mingay, S.: Bimodal IT: How to Be Digitally Agile Without Making a Mess, Gartner 2015, https://www.gartner.com/en/documents/2798217. Ultima visita Giugno 2024.

[Nexus2015] Nexus Guide https://www.scrum.org/Portals/0/NexusGuide%20v1.1.pdf, 2015. Ultima visita Giugno 2024.

[Patt2014] Patton, J.: User Story Mapping. O'Reilly, 2014

[Pich2010] Pichler, R: Make the product backlog deep. http://www.romanpichler.com/blog/make-the-product-backlog-deep/, 2010. Ultima visita Giugno 2024.

[PMI] PMI Project Management Institute. http://www.pmi.org/. Ultima visita Giugno 2024.

[Popp2003] Poppendieck, M.: Lean Software Development: An Agile Toolkit. Addison-Wesley Professional, 2003

[Rein1997] Reinerstsen, D. G.: Managing the Design Factory – A Product Developer's Toolkit. Simon & Schuster, 1997

[Ries2011] Ries, E.: The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses, Crown Publishing Group, 2011

[SAFe] SAFe – Scaled Agile Framework. http://www.scaledAgileframework.com. Ultima visita Giugno 2024.

[SatS] Scrum at Scale Framework: https://www.scruminc.com/scrum-incs-scrum-at-scale-framework/. Ultima visita Giugno 2024.

[Scrum2020] Schwaber, K. & Sutherland, J.: The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game, July 2020. https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf. Ultima visita Giugno 2024.

[ShYo2006] Sheppard, J. M.; Young W. B.: Agility literature review: Classifications, training and testing. Journal of Sports Sciences 24(9): 919–932, 2006

[SofS] Scrum of Scrums https://scrumguide.de/scrum-of-scrums. Ultima visita 2024, disponibile solo in tedesco.

[TaNo1986] Takeuchi, H.; Nonaka, I.: The new new product development game. Harvard Business Review 64(1), January/February 1986, p.137–146

[Wake2003] Wake, B.: Invest in Good Stories and Smart Tasks, http://xp123.com/articles/invest-in-good-stories-and-smart-tasks/. Ultima visita Giugno 2024.