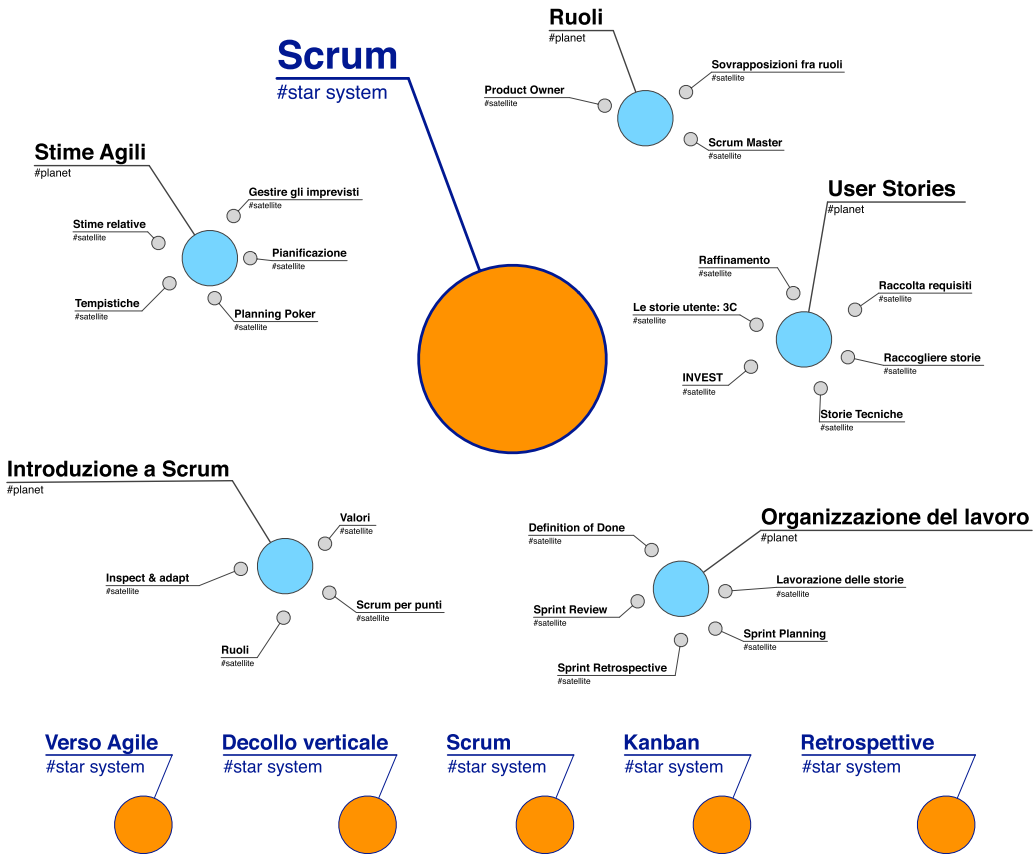
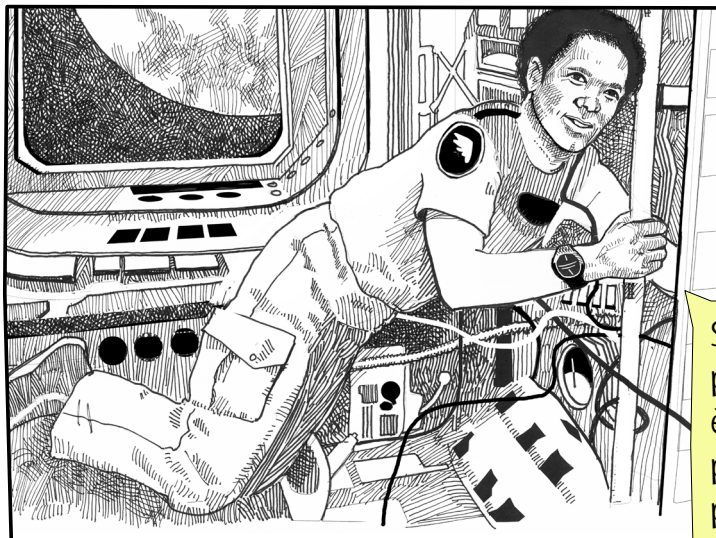
PARTE III
IL FRAMEWORK SCRUM

Giovanni Puliti



Parte 3
Il sistema Scrum

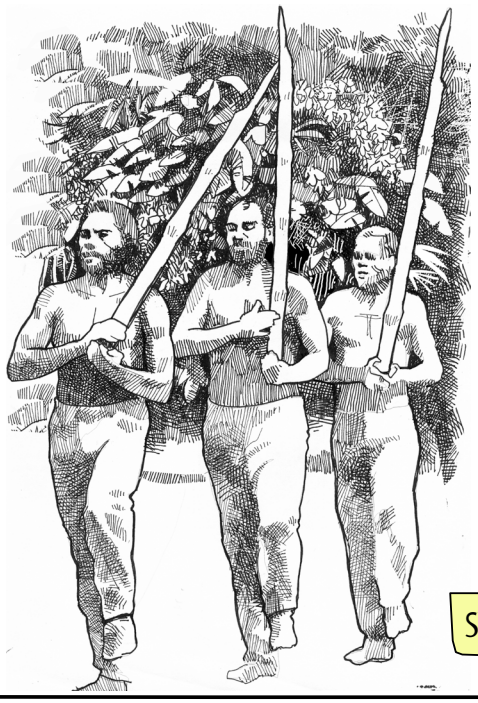


…Ramonek siamo arrivati?

Sì comandante, la procedura di preparazione del teletrasporto è terminata. Possiamo procedere al trasferimento sul pianeta Scrum.

Finalmente entriamo nel vivo del nostro viaggio. Anche se è da molto tempo che desidero visitare questo pianeta, non nascondo che solo adesso mi rendo conto di quanto siano state importanti le tappe precedenti.

È vero Ramonek. Come ti avevo detto, se non si parte prima dai principi e dai valori di Agile, il rischio è di imparare in modo meccanico delle pratiche senza nemmeno conoscere i veri motivi…



…ti ricordi del Cargo Cult?

Sì, certo comandante!

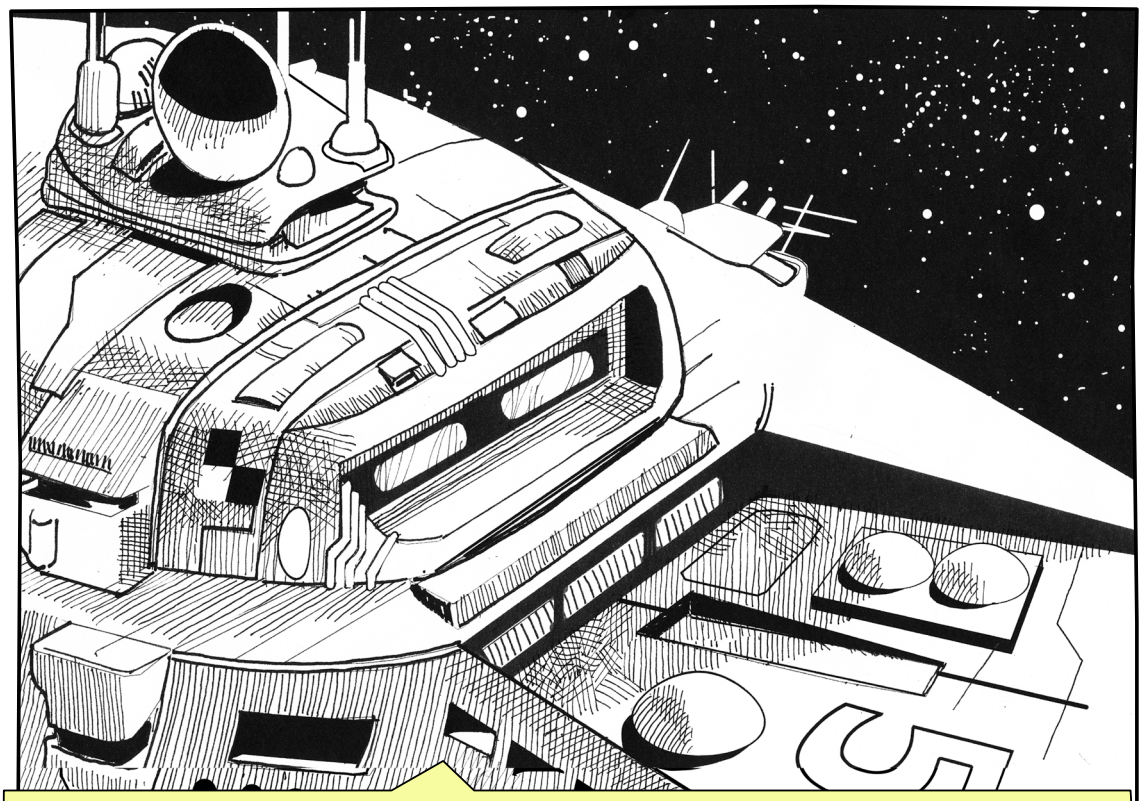
Ora che abbiamo visto cosa ha portato quel gruppo di agilisti a scrivere l'Agile Manifesto in quel famoso fine settimana sulla neve, possiamo passare a comprendere i dettagli tecnici delle metodologie agili, Scrum e Kanban.

E cosa vedremo su questo pianeta?

L'escursione che ho organizzato prevede prima una rapida panoramica dei concetti fondanti della metodologia: eventi, ruoli, artefatti.

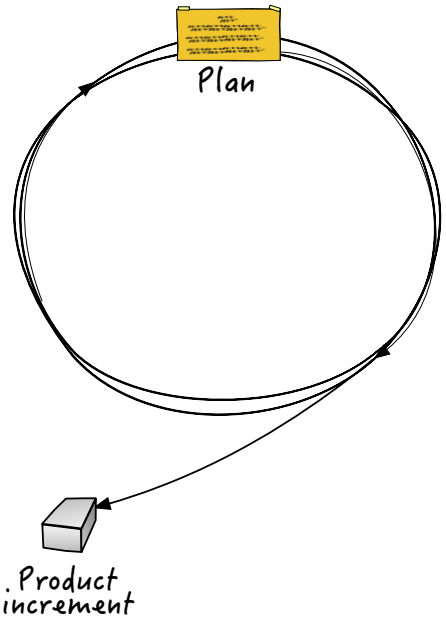
Scrum
star system

Nel nostro viaggio entreremo nel dettaglio di ognuno di questi aspetti.



Scrum definisce solamente poche regole da seguire… in questo senso è un framework il cui scopo è il miglioramento del gruppo.

In Scrum il lavoro è organizzato in iterazioni: a ogni iterazione il team rilascia qualcosa di finito che va a completare il prodotto finale.

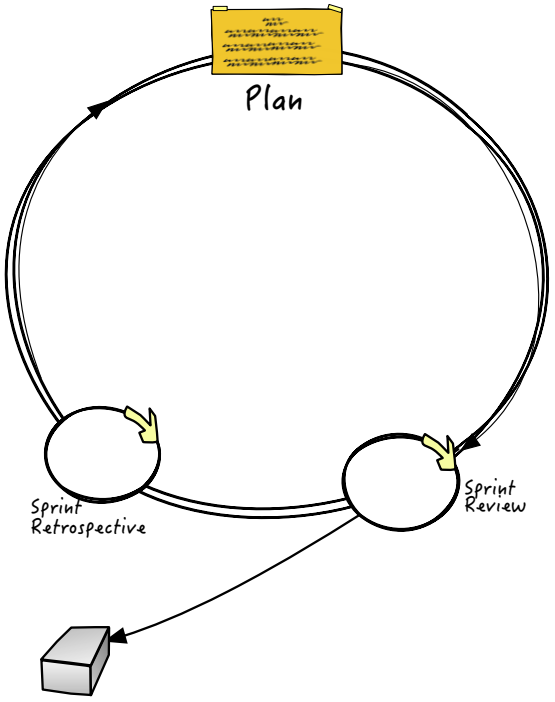


Per questo si lavora prima di tutto in modo da pianificare ogni iterazione tramite una riunione che si chiama Sprint Planning.

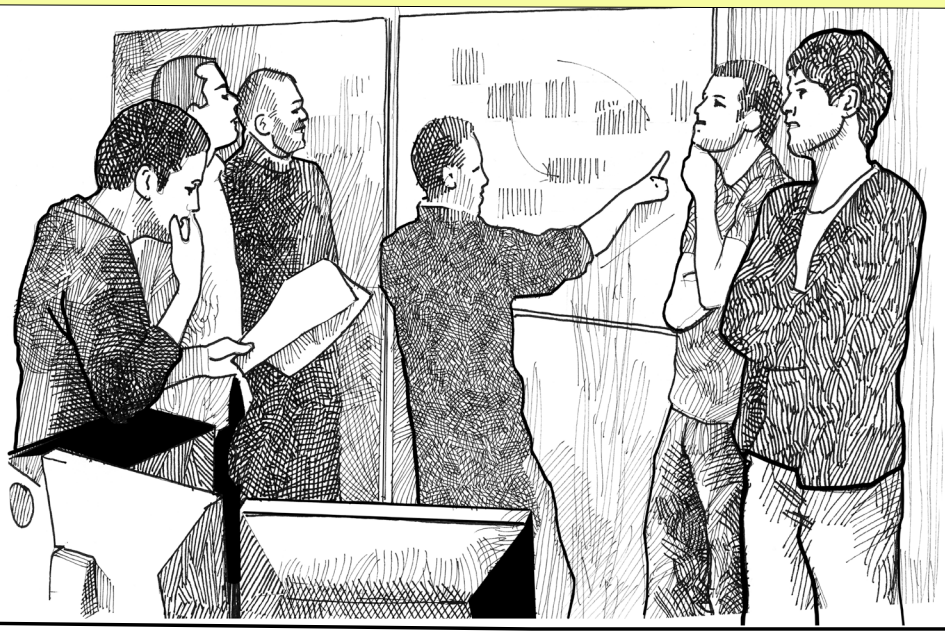
Dato che si segue un'approccio empirico, la pianificazione serve solo per dare un'impostazione di massima a quanto verrà fatto nell'arco di tempo dell'iterazione. Per questo è estremamente importante al termine di ogni iterazione verificare se le ipotesi fatte erano corrette.

Si verifica quindi sia la qualità del lavoro fatto, ma anche come lo si è fatto ossia si valuta il modo in cui il team ha lavorato.

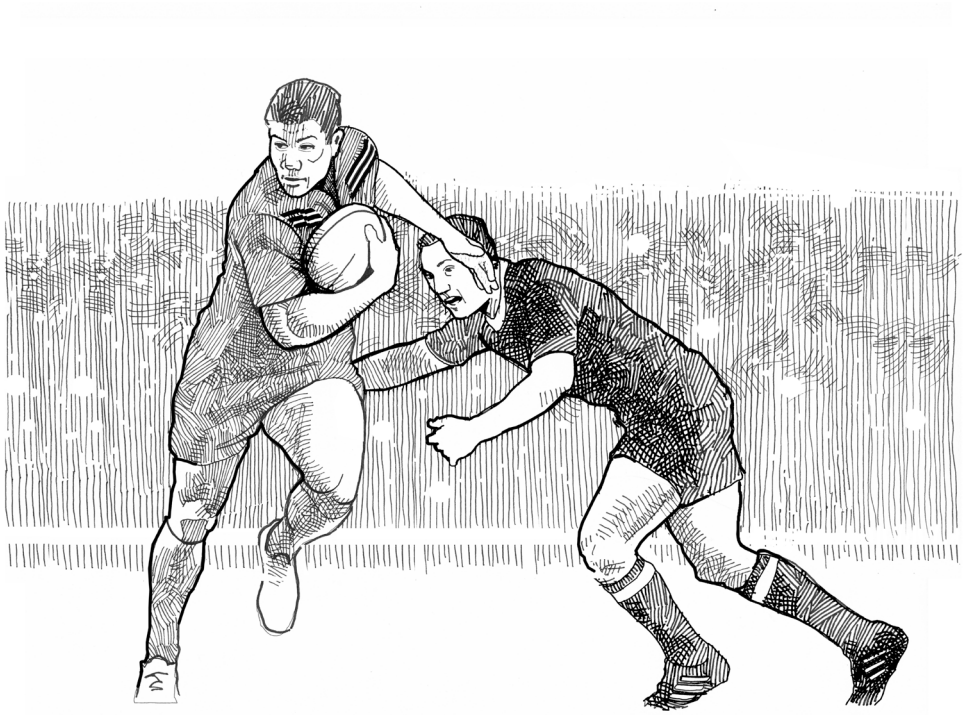
Queste verifiche si svolgono all'interno di altrettanti eventi detti Sprint Review e Sprint Retrospective.



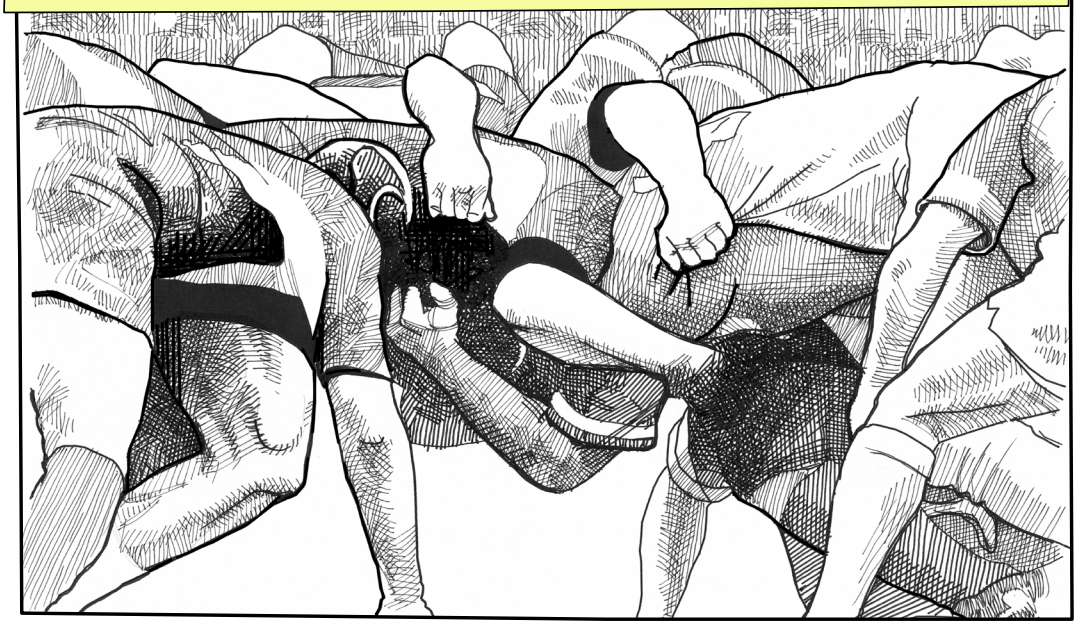
A queste sia aggiunge il Daily Scrum: una riunione quotidiana che si svolge in piedi (per questo è nota anche come Daily Standup Meeting) e che deve durare non più di 10'-15'. Serve per allineare i colleghi sul proprio lavoro e per condividere eventuali difficoltà.

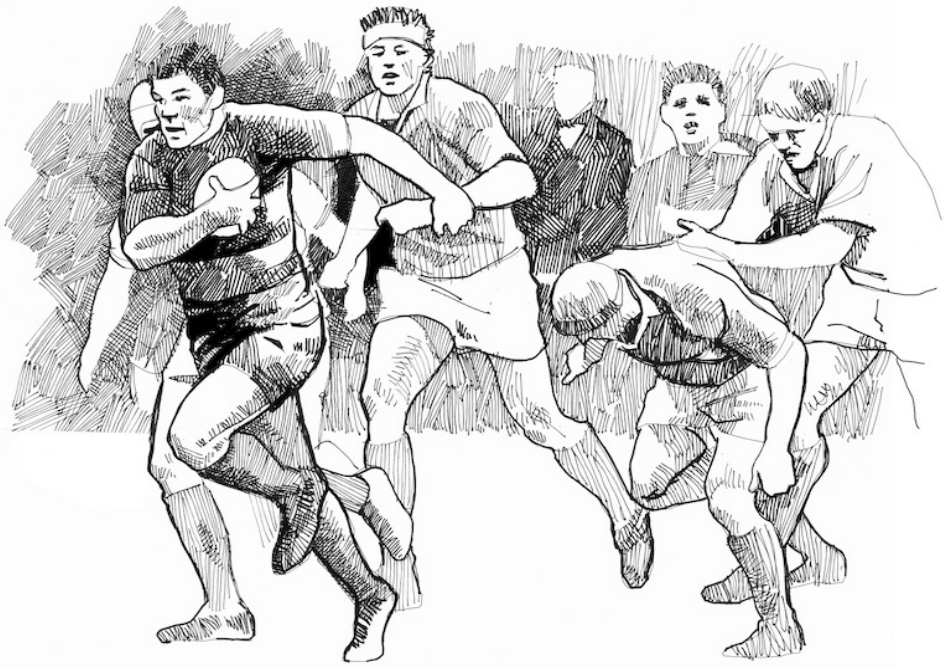


Comandante, lei ha citato spesso il termine Sprint. Cosa significa?



Il termine Scrum, che nel rugby è la mischia, è stato utilizzato per la prima volta a cavallo degli anni Novanta per descrivere un nuovo approccio allo sviluppo di prodotti in cui le persone sono concentrate insieme sullo stesso obiettivo: il successo del progetto.

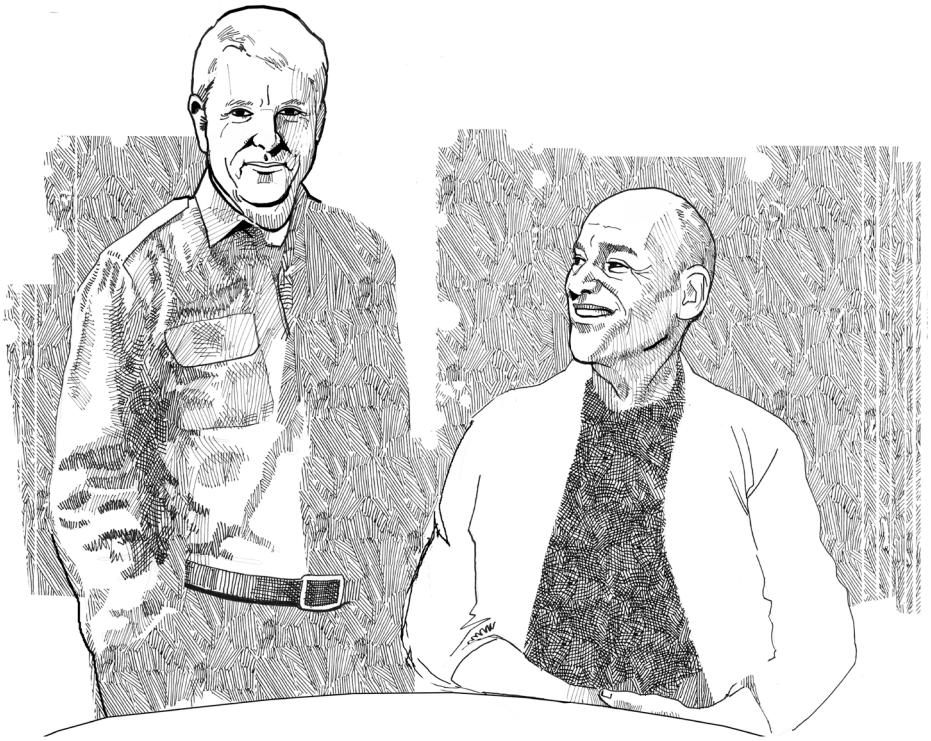




Nel rugby lo sprint è quel tratto di corsa che si fa per portare avanti il pallone. In Scrum, per raggiungere l'obiettivo, ogni membro del team guarda avanti, senza però perdere di vista i colleghi, proprio come nel rugby, dove il giocatore che corre verso l'area di meta deve sempre sapere dove sono i compagni di squadra a sostegno, per poter passare la palla.

Nel rugby, il giocatore con la palla corre in avanti seguito dai suoi compagni che lo sostengono: spirito di gruppo, comunicazione, empatia sono tutte caratteristiche importanti nel gioco del rugby così come in un team che deve sviluppare un prodotto software.

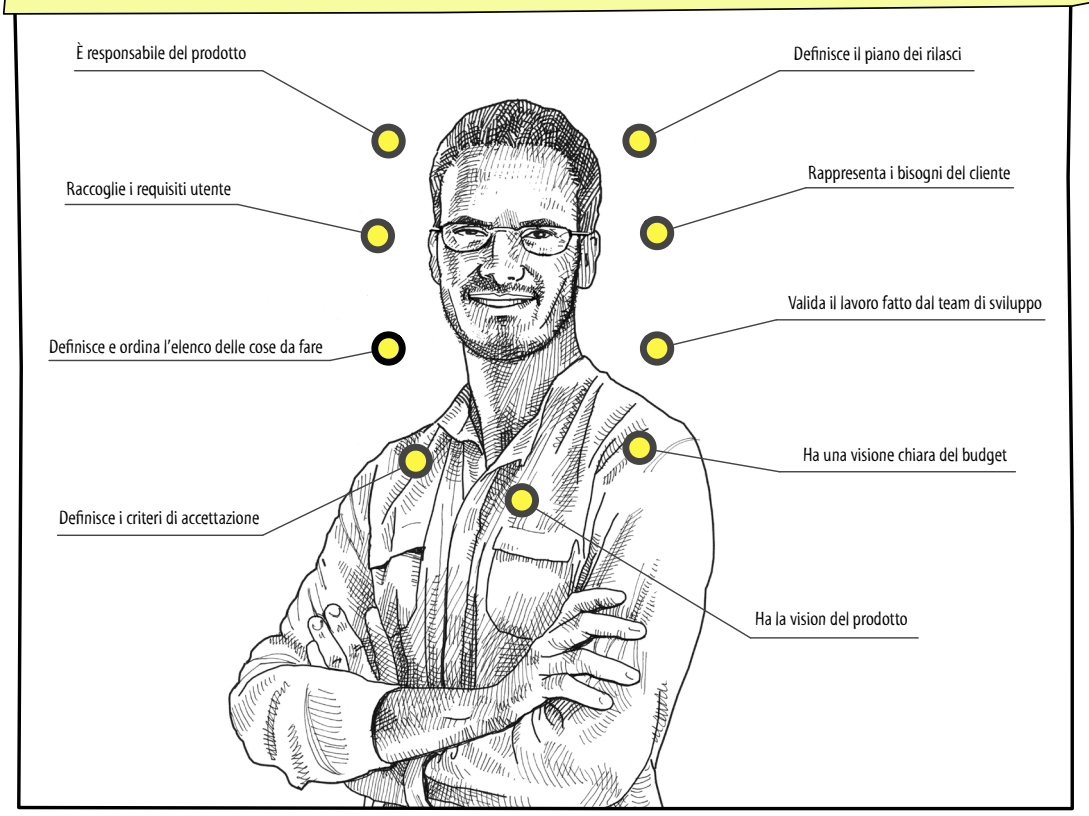
Ken Schwaber e Jeff Sutherland hanno presentato per la prima volta Scrum alla conferenza OOPSLA del 1995. Questa presentazione ha essenzialmente documentato ciò che Ken e Jeff avevano appreso negli anni precedenti dall'esperienza fatta sul campo.



Molto interessante comandante… E, riprendendo il discorso sugli elementi che costituiscono Scrum, oltre alle riunioni, accennava ai ruoli…

Sì, in Scrum si dà molta importanza ai ruoli che ci sono in un team: il Product Owner, lo Scrum Master e il team di sviluppo o Dev Team.

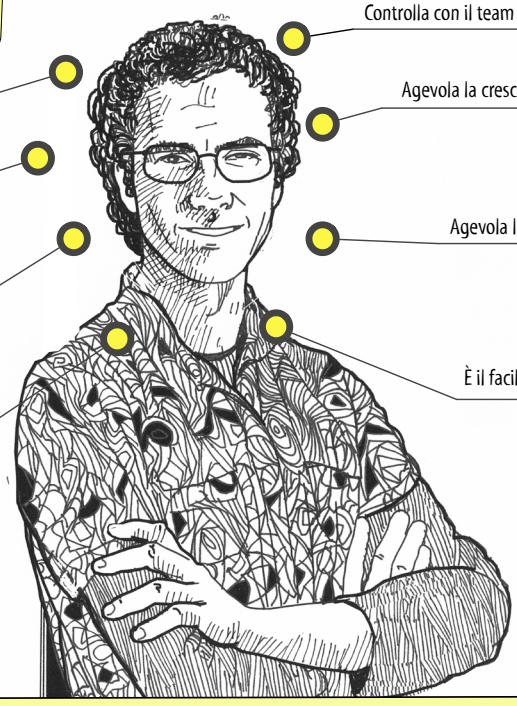
Il Product Owner, abbreviato in PO, è colui che ha la responsabilità del prodotto. Conosce l'utente e i suoi bisogni. È responsabile del cosa sarà contenuto all'interno del prodotto, ma non del come questo verrà realizzato, che è di competenza del team di sviluppo.



Il team di sviluppo (Dev Team) è composto da persone che dovrebbero avere tutte le competenze per sviluppare il lavoro. Per questo dovrebbero sempre avere la massima autonomia sulle scelte tecniche ossia sul come implementare le funzionalità.

Il PO quindi dice "il cosa", il Dev Team "il come".

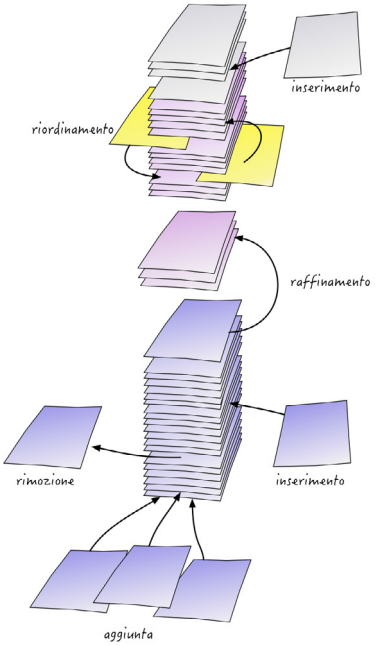
…e lo Scrum Master?



- Guida l'adozione di Scrum
- È il coach del team
- Fa da mediatore fra PO e Dev Team
- Rimuove gli impedimenti
- Controlla con il team gli stati di avanzamento
- Agevola la crescita delle persone nel team
- Agevola lo scambio di informazioni
- È il facilitatore dei vari meeting

Lo Scrum Master è un ruolo importantissimo per il team. La sua figura è spesso fraintesa come quella di "capo" del team (la parola master non aiuta); ma in realtà è un coach al servizio del gruppo. Aiuta a far crescere il team, agevola la comunicazione, fa in modo che si rispettino le indicazioni e molto altro ancora…

Comandante, cosa mi dice del terzo elemento fondamentale di Scrum, gli artefatti?

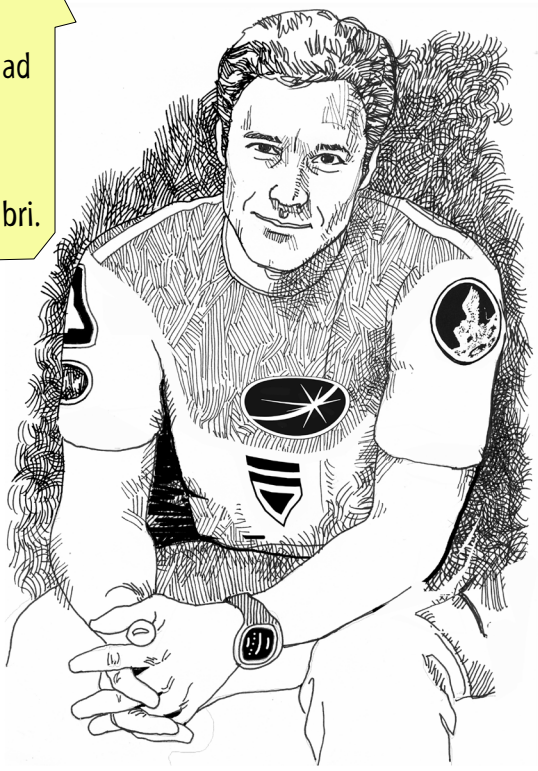


Avremo modo di vederli approfonditamente nel corso della nostra visita. Per adesso sappi che il Product Backlog, il Burn-Down Chart, e per certi versi anche la Task Board, sono importanti strumenti che aiutano il team a tenere sotto controllo il lavoro. Sono molto importanti, ma ricordati cosa è scritto nell'Agile Manifesto: persone e interazioni, piuttosto che strumenti e processi.

Comandante, da quello che mi par di capire, in Scrum ci sono molte regole, indicazioni: sembra una metodologia piuttosto articolata.

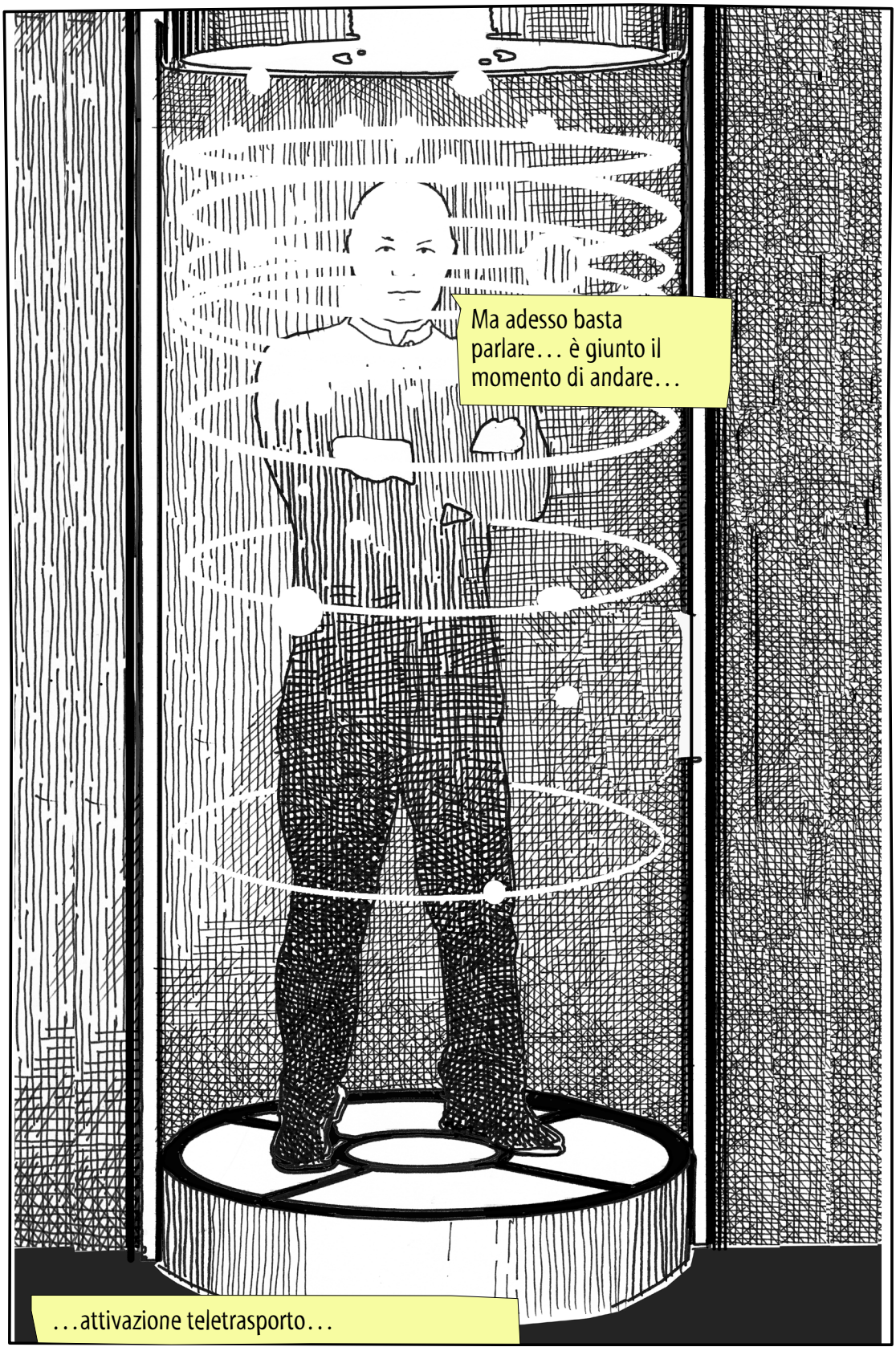
In realtà le regole e le prescrizioni sono poche e puntuali. Scrum può essere vista come una raccolta di best practice formulate sulla base di una lunga esperienza sul campo.

Probabilmente ogni situazione dove ti troverai ad agire sarà differente da quelle che trovi raccontate sui libri.

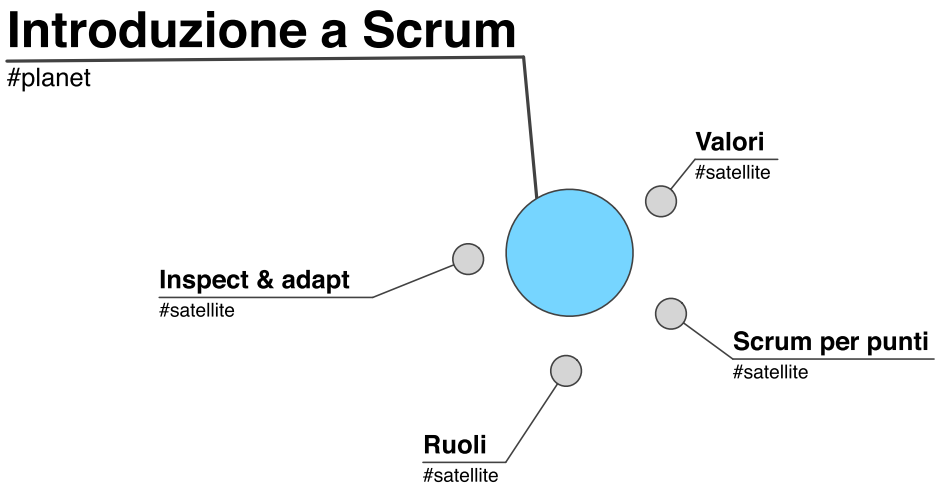


Partendo dai principi e dai valori di Scrum dovrai capire quando quelle raccomandazioni sono utili e quando invece potrai apportare qualche modifica.

Ricordati sempre dello Shu–Ha–Ri: ogni modifica a una disciplina dovrebbe essere apportata solamente quando possederai approfondite conoscenze dei fondamenti. Solo quando avrai fatto Shu (studio della teoria), Ha (apprendimento e radicamento dei principi e dei valori fondamentali), allora potrai fare Ri (modifica e personalizzazione della disciplina di base).


Ma adesso basta parlare… è giunto il momento di andare…
…attivazione teletrasporto…

Capitolo 1
Introduzione al framework Scrum



Che cosa è Scrum?

Scrum è un framework **di project management** per lo **sviluppo iterativo** e **incrementale di prodotti** ed è stato messo a punto per gestire progetti software. Gli inventori sono Ken Schwaber e Jeff Sutherland, i quali affermano [GS] che:

"Scrum è un framework di processo utilizzato dai primi anni Novanta per gestire lo sviluppo di prodotti complessi. Scrum non è un processo o una tecnica per costruire prodotti ma piuttosto è un framework all'interno del quale è possibile utilizzare vari processi e tecniche. Scrum rende chiara l'efficacia relativa del tuo product management e delle pratiche di sviluppo usate in modo da poterle migliorare."

Una celebre frase di Ken Schwaber dice che "Scrum non ti porta all'eccellenza, ma ti mostra il tuo livello di inefficienza". In questo senso Scrum deve essere considerato un sistema **diagnostico** con il quale individuare e comprendere i propri possibili punti di miglioramento.

Scrum è orientato alla gestione di progetti e per questo non si occupa delle eventuali ulteriori strutture necessarie per gestire l'organizzazione, anche se un uso corretto del framework porta inevitabilmente a una loro evoluzione: in questo senso, i tre ruoli tipici di Scrum per esempio (**Product Owner**, **Scrum Master** e **Dev Team**) sono più che sufficienti per sviluppare un prodotto.

Analogamente la parte della gestione del personale (stipendi, crescita professionale, carriera) non è di competenza del framework, ma di altre discipline e ruoli.

I valori di Scrum

Le tecniche e pratiche di Scrum si basano su **valori** che devono essere condivisi dai vari *stakeholder* di progetto, primo fra tutti il team: solo partendo da questi principi ed evitando l'applicazione meccanica e ripetitiva di regole e pratiche, si può ottenere il massimo potenziale da questo strumento.

I valori di Scrum sono **focalizzazione**, **coraggio**, **apertura**, ***commitment*** (nel senso di onorare l'impegno preso) e **rispetto**. Vediamo in cosa consistono.

Focalizzazione

Per lavorare bene, è necessario focalizzare la propria attenzione su **poche cose alla volta**. Concentrando i propri sforzi su pochi elementi per volta, migliorano la qualità complessiva e i tempi di consegna.

Coraggio

Quando si lavora in un contesto **complesso** non esistono soluzioni valide e pronte all'uso. **Provare e verificare** è la strategia vincente. Per questo è necessario prendersi la responsabilità delle proprie decisioni, consci del fatto che si potrebbe sbagliare completamente direzione. Coraggio di agire, coraggio di andare incontro al fallimento.

Importantissimo per questo è il lavoro di squadra in cui ritrovare supporto dei colleghi per affrontare i compiti più impegnativi.

Apertura

Elemento essenziale per rendere efficace e piacevole il lavoro di gruppo è la **trasparenza** circa le **motivazioni** e le azioni intraprese da parte di tutti i componenti del team: la **condivisione** di idee, dei problemi e delle soluzioni, dei successi e dei fallimenti, delle preoccupazioni, ma anche della fiducia nei propri mezzi e in quelli del team, sono il fondamento per la costruzione di un gruppo coeso e performante.

Apertura vuol dire inoltre accettare le proposte dei colleghi, accogliere nuove idee o strade alternative. Non sempre la soluzione a un problema è evidente, ma sono spesso necessari tentativi ed esperimenti.

Commitment

Per avere un controllo maggiore sulle nostre azioni e raggiungere gli scopi prefissi è necessario un costante **impegno** nell'applicazione dei principi di Scrum, nella verifica passo dopo passo delle conseguenze delle azioni intraprese (retrospettive), nel rispetto delle tempistiche (*time boxing*), dei ruoli e delle poche ma importanti regole del framework.

Rispetto

Lo spirito di gruppo deve basarsi su un forte **rispetto** del lavoro degli altri, del differente approccio al lavoro quotidiano e ai problemi che si incontrano, dei bisogni e aspettative delle persone. Rispetto vuol dire **accettare** le proposte degli altri, comprendere che non tutti abbiamo lo stesso punto di vista e che solo accettando quello degli altri, analizzandolo con obiettività il team potrà crescere. Un celebre adagio dice si dovrebbe essere duri con le azioni (nel senso di applicarvi una critica obiettiva e razionale), ma morbidi con le persone (la critica e le offese personali spesso non sono di alcun aiuto, ma anzi possono drammaticamente peggiorare il clima di lavoro). Lavorando insieme, si "vince" e si "perde" **insieme** e il rispetto è anche una conseguenza della presa di coscienza di questo fatto.

Il cambiamento è benvenuto

In Scrum poter gestire cambiamento è un aspetto fondante su cui si poggia l'intero framework: Scrum è ottimizzato per permettere all'organizzazione di **cambiare direzione** al progetto in maniera efficiente, non necessariamente per ottenere lo sviluppo più veloce possibile. Per questo motivo è importante che la mentalità delle persone sia in linea con questo modo di lavorare.

Generare valore, progressivamente, in maniera adattiva

Uno dei focus principali di Scrum è legato alla **produzione di valore** tramite la prioritarizzazione delle attività da svolgere in modo da rilasciare prima le cose di maggior interesse per l'utente finale.

Il valore viene quindi prodotto in modo progressivo grazie a un **approccio iterativo** e **incrementale**, privilegiando prima di tutto la velocità di adattamento del progetto o del prodotto alle esigenze del committente.

Scrum non si pone come obiettivo primario quello della riduzione degli sprechi, che anzi possono verificarsi tra una iterazione e l'altra e che sono accettati se questo garantisce una maggiore responsività del sistema di sviluppo.

In questo senso Taiichi Ohno (si veda la prima parte del libro) sapeva benissimo che la produzione di massa in serie e quindi la pianificazione stile waterfall, è senza dubbio più economica quando si devono produrre un gran numero di oggetti in modo ripetitivo. Per la produzione di piccole serie (o ancor più per la produzione di pezzi unici come nel caso di un prodotto software) ha più **senso economico** ottenere **una produzione flessibile**, dato che non avrebbe senso produrre più unità finite di quante realmente necessarie.

In Scrum si minimizzano i tempi dedicati alla pianificazione e all'analisi svolta prima che il progetto inizi, dato che si predilige un approccio empirico: il **coinvolgimento** del **Product Owner**, le periodiche **verifiche** sul lavoro fatto e sul come lo si è fatto, l'approccio iterativo che permette in ogni momento di correggere la rotta, sono le precondizioni necessarie per l'implementazione del cosiddetto **Inspect&Adapt** tanto caro a Taiichi Ohno.

L'utilizzo di uno strumento dinamico e adattabile come il **Product Backlog** (l'elenco delle cose da fare) non solo aiuta ma anzi abilita questo tipo di approccio: consente infatti di adattare lo sviluppo del prodotto per inserire nuove funzionalità non previste inizialmente, eliminando quelle cose che si rivelano non più necessarie o, più in generale, di modificare l'ordine dell'implementazione in base alle nuove esigenze che insorgono durante il progetto.

Il processo di costruzione del prodotto finale avviene quindi per piccoli passi in cui si limitano l'incertezza e gli effetti di eventuali errori apportando tutte le correzioni del caso.

Scrum in breve

Gli elementi fondamentali di Scrum sono l'organizzazione del lavoro in **iterazioni**, la presenza di un **team**, la separazione delle mansioni secondo **ruoli** ben precisi, l'organizzazione delle attività in **eventi** e l'uso di alcuni **manufatti** per gestire la lavorazione. In questo paragrafo si introducono questi concetti, rimandando ai successivi per gli approfondimenti del caso. Al tema della retrospettiva è invece dedicata un'intera parte del libro.

Un'infrastruttura leggera per la gestione di progetto

Scrum è definito come un *lightweight framework for project management*, le cui caratteristiche principali sono:

- Far **lavorare assieme business** e **IT** in maniera co-operativa: per esempio, come si avrà modo di vedere, il **Product Backlog Refinement** o lo **Sprint planning** sono

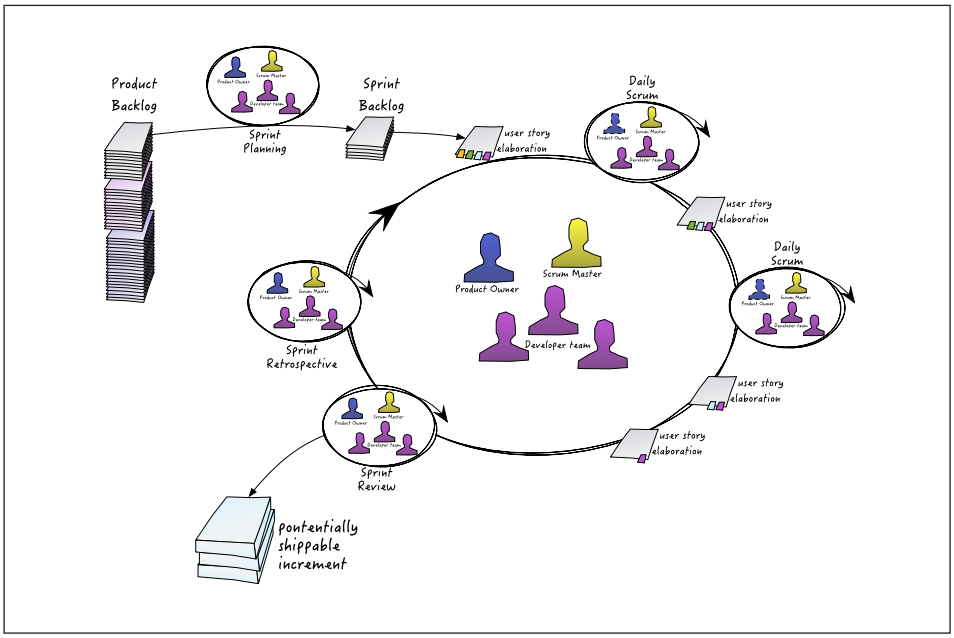


Figura 1. Diagramma di sintesi di Scrum, framework che si basa su ruoli, eventi e "artefatti".

attività fatte in modo collaborativo tramite sessioni di lavoro e di discussione di gruppo dal **Product Owner** e dal **team** di **sviluppo**.

- Approccio iterativo con **cicli di lavorazione** temporalmente ben definiti (*time control* invece di *scope control*).
- Trasparenza del processo (**Daily Scrum** e **Sprint Review**).
- **Pochi ruoli** con compiti e responsabilità **ben definiti**.

Vediamo meglio questi aspetti nei paragrafi successivi.

Processo iterativo incrementale

Per comprendere il funzionamento del framework, un buon punto di partenza è iniziare a comprendere come in Scrum si proceda verso la realizzazione del prodotto finito, ossia come si organizza il **calendario** e la **cadenza dei rilasci**.

Il lavoro viene suddiviso in **iterazioni** (dette **Sprint**), le quali rappresentano un lasso temporale ben preciso, con un **inizio** e una fine ben **definiti**.

Ogni iterazione comincia sempre con una **pianificazione** del **lavoro** che si vuole **eseguire all'interno dello sprint** e termina con una **verifica**, in cui si dà evidenza che il lavoro terminato rispetti gli standard di qualità prefissati.

Scrum pone massima attenzione alla stabilità dei **cicli** di lavorazione per garantire la regolarità delle consegne a fine sprint: più che inseguire il rispetto della consegna di tutto quanto previsto, è importante quindi non ritardare il termine dello sprint,

rimettendo quanto non terminato nell'elenco delle cose da fare, da eseguire magari in una delle iterazioni successiva, oppure mai più qualora non lo si ritenesse più necessario o utile.

È importante per questo motivo stabilizzare la **durata** dell'**iterazione** e la sua **cadenza** (giorno d'inizio e di fine sprint): è buona cosa in questo caso rispettare le abitudine e le esigenze del team il quale dovrebbe progettare l'organizzazione delle iterazioni in base alla proprie esigenze. A volte il team preferisce sincronizzarsi con il calendario settimanale (lo sprint comincia al lunedì e termina il venerdì) così da organizzare il proprio tempo in modo chiaro ed evidente; altri team lo considerano meno efficace perché in questo modo **Sprint Review** e **Sprint Retrospective** si fanno il venerdì cosa che potrebbe funzionare abbastanza male con l'avvicinarsi della pausa di fine settimana che può influire sullo stato d'animo delle persone del team.

Per quanto concerne invece la durata dello sprint, Scrum raccomanda di rimanere nell'intervallo che varia fra **una** e **quattro** settimane lavorative; citando direttamente la *Scrum Guide*: "The heart of Scrum is a Sprint, a time-box of one month or less".

In genere si lascia massima libertà al team che si auto-organizza nella scelta della durata funzionale al progetto e alla propria organizzazione. Dato che al termine di ogni iterazione si esegue una **verifica** del lavoro fatto, la lunghezza dello sprint incide sulla quantità di cose che il team realizza prima del controllo finale: maggiore è la durata dell'iterazione, maggiore sarà il rischio di trovarsi con implementazioni frutto di decisioni errate o semplicemente non efficaci; utilizzare **iterazioni brevi** permette di massimizzare il numero di verifiche e di ridurre il lasso di tempo che intercorre fra una verifica e l'altra.

Sebbene non ci sia una regola, nella maggior parte dei casi i team optano per sprint di **due settimane**, che è un buon compromesso fra l'avere sprint piccoli e limitare il costo del processo: organizzare eventi e tutte le attività di processo può avere un impatto più elevato su sprint di durata minore.

Gli eventi

In Scrum sono previste una serie di attività ricorrenti detti **eventi**: lo **Sprint Planning**, la **Sprint Review**, la **Sprint Retrospective** e il **Daily Meeting**. Si tratta di vere e proprie colonne portanti del framework, indispensabili per il processo stesso: senza un **Planning** non si parte, senza una **Review** non si può sapere se quanto fatto è corretto, senza la **retrospettiva** non si possono individuare, e quindi correggere, gli eventuali errori nel processo.

Lo Sprint Planning

Ogni iterazione comincia con la **pianificazione** di cosa sarà fatto nello Sprint. L'attività inizia con la presentazione delle necessità e priorità del business da parte del **Product Owner** (PO), una delle figure più importanti di Scrum: egli è il responsabile del risultato finale, e dirige il lavoro del team in modo da realizzare un prodotto che **soddisfi**

i bisogni dell'utente (che in Agilità è una delle definizioni di qualità del prodotto). Il Product Owner parla per conto del cliente e dell'utente, qualora non siano la stessa persona, ed è in grado di interpretarne i bisogni e presentarli in modo corretto al team.

Allo Sprint Planning il Product Owner quindi si presenta con una lista di **funzionalità da implementare** — vedremo meglio il formato e il contenuto di questi elementi della lista, per adesso possiamo genericamente chiamarli **items** — e le presenta al team seguendo fedelmente l'ordine di tale lista; come avremo modo di vedere l'ordinamento è funzionale alle scelte del PO per soddisfare le necessità degli utenti.

Per ogni funzionalità proposta, i membri del team, tramite una serie di domande e di indagini, cercano di ricavare il maggior numero di informazioni per poterne fare una valutazione circa il carico di lavoro necessario per la sua implementazione. Il team quindi decide se accettarla nell'elenco delle cose da fare all'interno dello sprint. Questo processo di accettazione procede fintanto che il team ritiene di aver raggiunto la **capacità massima** di **lavorazione** per lo sprint corrente: il Product Owner in tal senso non ha alcun potere per forzare il team di sviluppo ad accettare un numero maggiore di items. Quella appena descritta è la versione schematica del Planning: nella realtà le cose si organizzano in modo leggermente differente.

Il lavoro quotidiano: il Daily Scrum

Una delle attività che si svolgono regolarmente all'interno dello Sprint è il **Daily Scrum**, **riunione quotidiana** in cui il team si ritrova per parlare del lavoro fatto il giorno prima e per condividere eventuali difficoltà o punti di attenzione. La riunione ha una durata breve, normalmente **10-15 minuti** e si svolge **in piedi** (per questo a volte si chiama **Daily Standup Meeting**), per mantenere alta l'attenzione e per evitare che qualcuno si distragga leggendo mail o parlottando con un collega. Ogni membro del team espone tre "argomenti" agli altri: cosa **ha fatto** il giorno precedente, cosa **pensa di fare** il giorno corrente e se si sono incontrate delle **difficoltà** particolari nello svolgimento del lavoro del giorno precedente.

Vedremo in seguito che spesso i team Scrum prendono in prestito dalla metodologia **Kanban** l'uso di una lavagna fisica (la **task board** appunto) utile per segnare lo stato di avanzamento della lavorazione dei vari elementi. In questo caso, il **Daily Scrum** è il momento in cui le persone **aggiornano** questa board, segnalando eventuali impedimenti apponendo, ad esempio, un particolare simbolo sul cartellino corrispondente, oppure semplicemente spostano nella colonna delle cose terminate i cartellini delle attività finite.

Il Daily Scrum è quindi prima di tutto un momento importante di **allineamento** all'interno del gruppo: serve per condividere cosa sta succedendo nel progetto, per sapere cosa stanno facendo i vari colleghi del team, per informare delle problematiche incontrate e, nel caso, per proporre o condividere eventuali soluzioni.

È importante non "sforare" il tempo prefissato per questa attività. Per esempio, lo scopo del meeting **non** è quello di entrare nel **dettaglio** dei singoli impedimenti alla

ricerca di possibili soluzioni, ma piuttosto quello di condividere i problemi per poi pianificare una qualche iniziativa come un breve momento di studio mirato da fare in coppia, l'esecuzione di un test o anche semplicemente uno scambio di pareri con un collega che ha avuto modo di gestire un problema analogo.

Data la sua semplicità, il rapporto costo/benefici è certamente a favore di questa pratica: il Daily Scrum infatti è introdotto sia in quelle organizzazioni che intendono applicare Scrum in modo preciso e rigoroso, sia dove ci sia semplicemente l'interesse di applicare un qualche **processo di miglioramento**, indipendentemente dal fatto che si faccia Scrum o meno.

Al **Daily Scrum** in genere partecipa tutto il team di sviluppo e anche lo **ScrumMaster** il quale svolge il ruolo del facilitatore.

Ufficialmente la presenza del **Product Owner** non è richiesta in questa riunione; nella pratica la sua partecipazione è utile sia per il PO che per **Developer Team**: il primo rimane allineato sullo stato di avanzamento del lavoro e su eventuali impedimenti, i secondi possono sapere cosa il PO sta facendo o cosa si appresta a fare o anche semplicemente se sta incontrando particolari impedimenti.

La Sprint Review

Al termine di ogni sprint, per essere certi che il lavoro fatto sia non solo **funzionante**, ma anche **corrispondente** alle esigenze dell'utente, si esegue una dimostrazione del lavoro fatto, tramite un incontro che si chiama **Sprint Review**. A questa riunione il Product Owner partecipa con il ruolo di validatore finale: è lui che comunque interpreta i giudizi degli utenti o committenti, anche se può essere utile la presenza de gli utenti finali che possono contribuire alla valutazione del lavoro fatto dal team di sviluppo.

Le attività che non passano la **Sprint Review** saranno reinserite in lista per una lavorazione successiva. Sarà compito del Product Owner decidere se e quando tale lavorazione debba essere eseguita: potrebbe essere alla iterazione immediatamente successiva, oppure se ne potrebbe ritardare la lavorazione in attesa che maggiori informazioni siano raccolte per risolvere i problemi insorti in fase di Sprint Review; oppure, ancora, il PO potrebbe decidere che tale funzionalità non debba più essere implementata e in quel caso viene rimossa dal backlog.

Nelle prime versioni di Scrum, la **Sprint Review** era considerata "il" momento di verifica in cui validare la correttezza e quindi l'accettazione delle attività svolte. Man mano che il team completava le storie, queste erano parcheggiate in attesa della verifica di fine sprint. Col tempo, pur continuando a considerare la cerimonia dello **Sprint Review** uno dei momenti più importanti dell'intero sprint, si è iniziato a valutare l'ipotesi di valutare le funzionalità man mano che il team ne completa l'implementazione.

In questo modo si può ridurre l'incertezza del risultato finale, non essendo necessario aspettare la fine di uno sprint per verificare se il lavoro fatto soddisfi i desideri dell'utente o del Product Owner; è questo un ottimo modo di ridurre il rischio, minimizzando il numero di funzionalità implementate ma non ancora verificate. Inoltre, anticipando

la verifica delle attività svolte prima della demo finale, si può distribuire l'effort legato al lavoro di controllo: in questo modo la demo finale finisce per essere poco più di una rapida formalità.

La valutazione del lavoro svolto quindi non dovrebbe mai essere un'attività dell'ultimo minuto, ma i feedback dovrebbero essere raccolti man mano che il team di sviluppo completa qualcosa: prima che lo sprint termini vi è quindi il tempo sia per correggere gli errori trovati nelle funzionalità implementate, o impedire che eventuali errori si propaghino anche sulle altre cose non ancora terminate.

NOTA: Il nome ufficiale attuale di questa attività è **Sprint Review**, anche se nel corso degli anni ci sono state diverse varianti e interpretazioni sull'esatto svolgimento, nonché sul nome: per un periodo di tempo ha preso campo il termine **Sprint Demo**. A un certo punto si è provato a dividere questo meeting in due parti separate: una prima di presentazione delle attività svolte nello sprint, descrivendo gli obiettivi prefissati, le difficoltà incontrate o viceversa i risultati raggiunti, e una seconda di verifica vera e propria. Attualmente la *Scrum Guide* parla solo di **Sprint Review** come del momento in cui si effettua la verifica finale delle funzionalità completate. Nella pratica spesso è ritenuto utile far precedere l'attività di verifica da una breve introduzione e spiegazione del lavoro fatto in relazione agli obiettivi di sprint.

La Sprint Retrospective

Al termine di ogni iterazione il team Scrum esegue una valutazione non tanto del **cosa** è stato fatto, che è oggetto della **Sprint Review**, ma piuttosto del **come** il team ha lavorato, delle difficoltà incontrate, del come i problemi sono stati affrontati e risolti.

In questa fase non si analizza il **prodotto** ma piuttosto il **processo**. Questa attività in Scrum si chiama **Sprint Retrospective**, della quale si parla in modo più dettagliato nella parte quinta di questo libro.

È grazie alle retrospettive, eseguite con regolarità e puntualità, che il team è in grado di migliorare il proprio lavoro e di diventare sempre più performante. Molteplici sono le tecniche e gli strumenti a disposizione del team per indagare sui problemi incontrati o per mettere in evidenza le necessità del team. Nella parte dedicata alle retrospettive, ne sono illustrati principi e sono presentati i diversi approcci, in modo approfondito e aggiornato.

Raffinamento

Un elemento che non è ancora stato introdotto è il **Product Backlog**, una specie di "pila" ordinata contenente le funzionalità che il team deve implementare per rispondere alle esigenze dell'utente, ossia per completare il prodotto. Questa pila conterrà in alto elementi sufficientemente elaborati per essere passati al team di sviluppo che provvederà alla lavorazione. Nella parte bassa della "colonna", invece, gli elementi della pila sono ancora molto grezzi, ossia sono poco più che titoli di macro funzionalità.Il team

non sarebbe in grado di procedere all'implementazione perché tali elementi sono ancora troppo grossi e scarsamente dettagliati.

Il processo di trasformazione delle "cose grosse e poco dettagliate" in "funzionalità pronte per essere messe in lavorazione" si chiama appunto **raffinamento** (o **Product Backlog Refinement**): è un'attività seguita e coordinata dal Product Owner che si avvale di volta in volta delle persone interessate o disponibili del team di sviluppo; in genere viene svolta in modo libero e senza che sia seguita una qualche tempistica, ma in base alle necessità del progetto. Lo scopo è permettere che al planning successivo ci siano abbastanza funzionalità pronte per essere messe in lavorazione da parte del team di sviluppo. Durante il raffinamento del backlog, nuove funzionalità appaiono nel backlog sia come risultato del processo di scomposizione delle macro-funzionalità grezze, sia a causa dell'inserimento di nuove funzionalità frutto di nuove scoperte. Altre invece potranno essere eliminate o spostate di ordine (figura 3).

Il **Refinement** è un momento in cui business e team collaborano insieme, come del resto in molti altri momenti, e per questo motivo può essere visto come l'implementazione di uno dei principi dell'Agile Manifesto.

NOTA: Il termine **Refinement**, o raffinamento, sta prendendo gradualmente il posto di quello originario **Grooming** che nella sua definizione originaria indica l'operazione di "spulciamento" che alcuni animali (anzitutto le scimmie, ma anche insetti sociali come le api) svolgono per individuare eventuali parassiti sul corpo dei compagni e rimuoverli. Un altro significato del termine è la strigliatura dei cavalli o il lavoro di preparazione di un cane/gatto per una mostra, dove il padrone pettina e pulisce bene il pelo dell'animale per presentarlo al meglio ai giudici.
Ma, nell'inglese britannico gergale, il termine *grooming* indica anche le pratiche che i pedofili mettono in atto per adescare le loro vittime: è per questo significato equivoco che tale parola sta gradualmente scomparendo dal gergo Scrum dove rimane solo *refinement*.

Ruoli in Scrum

Il framework prevede la definizione di tre ruoli: il **Product Owner**, lo **ScrumMaster**, il **Dev Team**. Questi tre ruoli insieme formano lo **Scrum Team** (figura 2).

Product Owner

Il **Product Owner** è il responsabile per il massimo valore possibile dal prodotto e dal lavoro del team di sviluppo: in una parola si potrebbe dire che si prende cura del ROI del prodotto.

Per fare questo egli conosce le funzionalità che dovranno comporre il prodotto finale, conosce quindi le esigenze di business dell'utente. Egli ha la responsabilità ultima di decidere **cosa** deve essere fatto, ma non di stabilire **come**. Per questo motivo è il responsabile del contenuto e del ciclo di vita del **Product Backlog**, del quale coordina e guida ogni attività di raffinamento, inserimento di nuove storie, modifica o rimozione

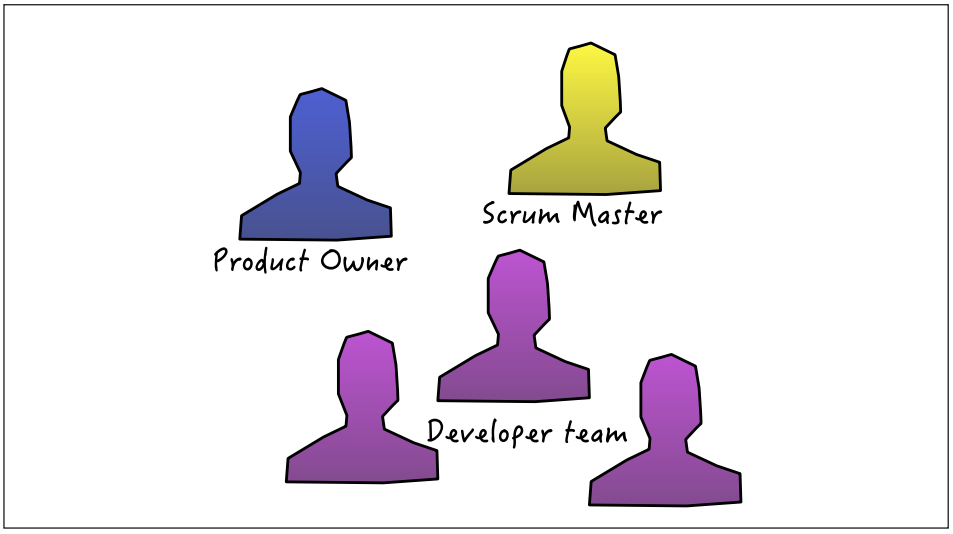


Figura 2. Un team Scrum è composto da un Product Owner, uno ScrumMaster e dal Dev Team.

di quelle esistenti. Il tema della Product Ownernship è molto importante per la buona riuscita del progetto: un **Product Owner** debolmente coinvolto nel progetto, con poca attenzione o dedizione al progetto, che non abbia una vision chiara o non sappia trasmetterla in modo coerente può costituire un grosso rischio per la buona riuscita del progetto.

Dev Team

Il **Development Team** è il gruppo di sviluppo che svolge il lavoro di implementazione. Lavora a stretto contatto con il **Product Owner** — ma anche con tutti gli altri potenziali stakeholder — per comprendere al meglio i dettagli dell'implementazione. Decide **come** implementare — scelte tecniche, architettura, dettagli implementativi — ma non entra nel merito del **cosa** debba essere realizzato. **PO** e **Dev Team** collaborano secondo un approccio co-operativo, dove è più importante collaborare per il bene del progetto piuttosto che far valere il proprio ruolo. Il team di sviluppo per esempio può, anzi deve, discutere con il PO nel caso in cui certe scelte sub-ottimali dal punto di vista implementativo potrebbero essere migliorate se il prodotto fosse realizzato in un altro modo.

Nel caso in cui non si trovi un accordo o, peggio ancora, si arrivi a uno scontro di vedute, allora valgono le regole "dettate" da Scrum: **PO** stabilsce **cosa**, **Dev Team** sceglie **come**; ma arrivare a tal punto è comunque una sconfitta per tutti.

ScrumMaster

Lo **ScrumMaster** ha il compito di fare in modo che tutti i partecipanti al progetto possano svolgere al meglio il loro lavoro. Il suo compito è di fare da **coach** del team

[AC], di supportare o agevolare la risoluzione di ogni impedimento e di ridurre i rallentamenti.

Una definizione molto usata è quella di **servant leader**, anche se per certi versi questa può dare una visione distorta. Esiste anche una proposta alternativa secondo cui lo **ScrumMaster** è visto come un **host leader** [HL]: in questo caso il suo ruolo è paragonabile a quello di una persona che organizza una festa a casa propria e che quindi si preoccupa che tutto sia al suo posto in modo che tutti gli invitati possano divertirsi. Sarà accogliente e amichevole ma al tempo stesso è colui che si preoccupa che tutti gli invitati partecipino alla festa nel rispetto di quelle regole basilari necessarie per la buona riuscita della festa stessa. In questo senso lo **SM** è l'owner del processo, ossia colui che verifica che si seguano le regole definite da Scrum.

Gli artefatti

Gli artefatti di Scrum sono degli strumenti "manufatti" per consentire la gestione del lavoro che il team svolge per implementare il progetto. Il loro scopo è quello di comunicare nel modo più trasparente possibile sia l'elenco delle cose che devono essere realizzate sia lo stato di avanzamento del progetto stesso. Essi sono fondamentalmente il **Product Backlog** e lo **Sprint Backlog**.

A questi si aggiungono altri strumenti come il **Burn-Down Chart** e la **Task Board** che vengono però menzionati come "indicatori dell'avanzamento" (*Progress Indicators*); Scrum non li mette **ufficialmente** fra gli strumenti di processo, lasciando al team la valutazione se usarli o farne a meno.

Il Product Backlog

In Scrum, il prodotto finale è realizzato in modo **incrementale**, implementando una serie di funzionalità che sono inserite in una **lista ordinata** per **priorità** in modo da dare precedenza alle funzionalità di maggior valore, che verranno quindi implementate per prime. "Valore" può essere inteso come il valore economico, la risoluzione dei bisogni utente più urgenti, l'acquisizione di nuove conoscenze per il team, oppure la risoluzione dei rischi.

Questa lista prende il nome di **Product Backlog** e ogni elemento che la compone si chiama Product Backlog **Item**. Gli elementi che stanno in alto in questa lista sono quelli a cui viene data la **precedenza** nella lavorazione e per questo sono sufficientemente piccoli e dettagliati per permettere al team lavorarli nell'arco di uno sprint.

Man mano che si scende verso il basso nel **Product Backlog**, aumenta la dimensione e diminuisce il livello di dettaglio degli elementi contenuti: il processo di **raffinamento**, eseguito sugli elementi del backlog, infatti, ha lo scopo di preparare gli item prossimi alla lavorazione nello sprint — quelli in alto nella pila — rimandando la preparazione di quelli in basso al momento in cui ne sarà necessaria la messa in lavorazione e non prima. In un prossimo paragrafo si entrerà nel dettaglio del processo di alimentazione del **Product Backlog** e della modalità di lavorazione sui suoi elementi.

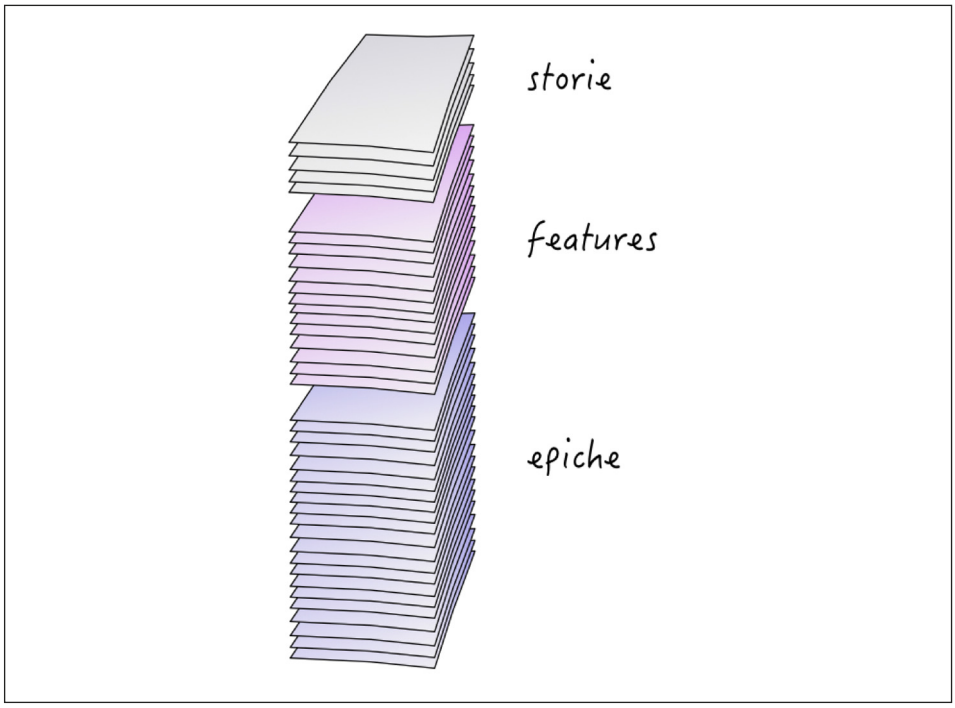


Figura 3. Il Product Backlog (PB) è una pila di elementi che verranno implementati dal team. In alto, gli elementi già raffinati sono della dimensione giusta per permetterne la lavorazione. In basso, invece, elementi più grossi rappresentano funzionalità o parti del sistema a grana grossa.

Scrum non impone alcun vincolo sul formalismo da utilizzare per la descrizione degli item del backlog, anche se nella maggior parte dei casi si utilizza il formalismo delle **User Stories**, di cui parleremo dettagliatamente più avanti. In questo caso gli item sono classificati secondo tre ordini di grandezza:

- **storie utente**, che sono sufficientemente piccole e dettagliate per essere messe in lavorazione da parte del team;
- **features**, ossia elementi più grossi che possono essere equiparati a un insieme di storie, per esempio tutte le funzionalità incluse in una finestra web;
- **epiche** vale a dire raccolte macroscopiche di funzionalità, corrispondenti per esempio a un menu di primo livello o a una unità logica di business dell'applicazione.

Da un punto di vista strutturale **storie**, **features** ed **epiche** sono la stessa cosa, cambia solamente la dimensione.

Da notare che spesso le definizioni sono usate in un ordine leggermente differente: si tratta sempre di convenzioni; in questo libro seguiremo lo schema appena descritto e illustrato in figura 3.

Sprint Backlog

Accanto al Product Backlog, che rappresenta il totale delle cose che devono essere realizzate per completare il prodotto, il team utilizza un altro strumento, detto **Sprint Backlog**, che contiene invece l'elenco delle cose che il team di sviluppo si impegna a implementare nell'arco di **uno sprint**. Anche in questo caso si tratta di una lista ordinata di attività da implementare che i vari membri del team, in modo totalmente auto-organizzato, mettono in lavorazione in base all'ordine del backlog e alle capacità delle persone.

A differenza del backlog di prodotto, il cui contenuto è in continua evoluzione e modifica, il processo di alimentazione dello **Sprint Backlog** segue una logica completamente differente. Il contenuto viene **fissato** a inizio sprint, durante la cerimonia dello **Sprint Planning** e non dovrebbe più essere modificato per tutto il resto dello sprint.

Ogni alterazione dello **Sprint Backlog** ha forti ripercussioni sul lavoro del team di sviluppo, sulla sua capacità di auto-organizzarsi il lavoro, nonché sulla abilità di stimare la propria capacità produttiva. Ci sono però casi in cui una certa flessibilità non solo è auspicabile, ma perfino necessaria. Si pensi al caso in cui, pur avendo inserito nello **Sprint Backlog** un elemento per la lavorazione, si scopra che esso non sia più necessario o, peggio, sia stato formulato in modo errato. Se non è ancora stato messo in lavorazione, converrà rimuoverlo dallo **Sprint Backlog**, anche se questo porterà certamente una perturbazione nel ciclo di lavorazione: le conseguenze di una non cancellazione sarebbero comunque peggiori. Conviene quindi accettare le implicazioni di tale variazione, ma al contempo interrogarsi, in fase di retrospettiva, sui motivi di questo evento.

Altra eccezione alla regola di non modificare lo **Sprint Backlog** si verifica quando il team di sviluppo termina la lavorazione su tutti gli elementi presi in carico all'interno dello sprint: in questo si può procedere alla presa in lavorazione di altri item direttamente dalla testa del **Product Backlog** senza la necessità di passare da una fase di planning; a fine sprint si avrà una percezione alterata della capacità di lavorazione del team: l'alternativa, però, sarebbe stata quella di bloccare il lavoro, per cui si accetta volentieri questa alterazione.

Burn-Down Chart

Un elemento che non è parte ufficiale della specifica Scrum ma che comunque molto utilizzato dai team è il **Burn-Down Chart**, un grafico che permette di monitorare lo stato di avanzamento del lavoro del team di sviluppo. Si tratta di un istogramma dove l'asse orizzontale rappresenta il tempo raffigurato in forma di step discreti (numero di sprint di progetto nel caso di un **Project Burn-Down**, giorno di sprint se si sta parlando di **Sprint Burn-Down**), mentre sull'asse verticale si riporta, step dopo step il numero di cose che devono **ancora** essere implementate.

Per realizzare un Burn-Down Chart si comincia segnando sull'asse verticale il **totale** delle cose da implementare (**punto A**), espressa in punti del backlog, e sull'asse orizzontale la **data attesa** di termine dei lavori (**punto B**).

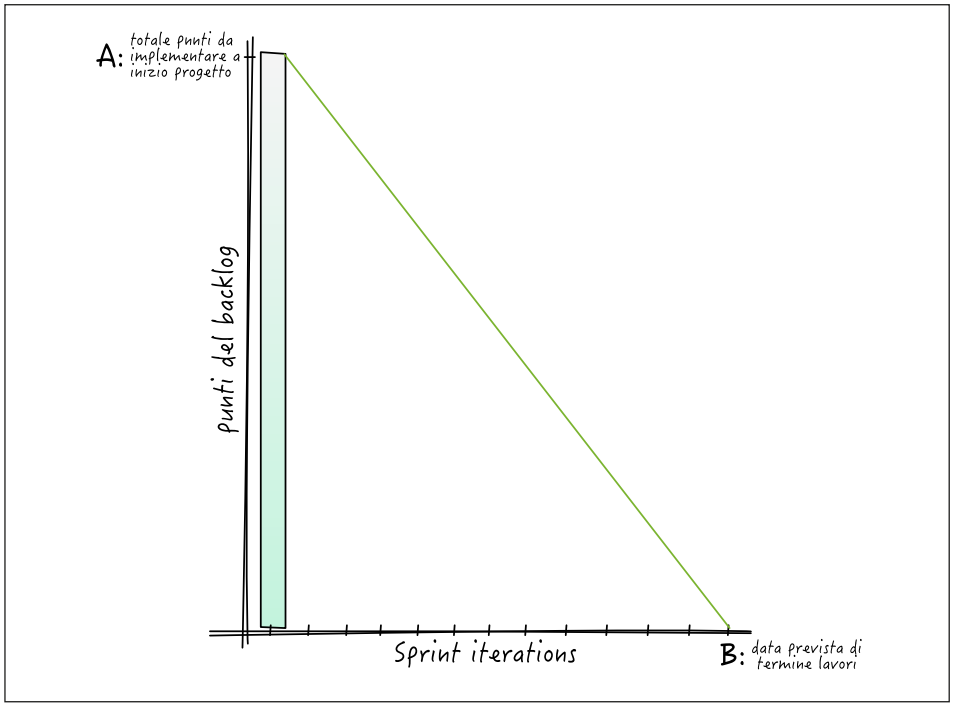


Figura 4. Il Burn-Down Chart, nella sua fase iniziale.

Tracciando la retta discendente che unisce A e B si può avere un'idea, iterazione per iterazione, di quale dovrà essere il **trend** dell'**implementazione** delle **storie** affinché si possa completare il progetto nel tempo prefissato. Tale retta a volte viene detta **trend a zero** (figura 4).

Al termine di ogni iterazione si disegna una nuova colonna sottraendo dalla colonna precedente la somma delle storie completate con successo nello sprint appena terminato. Dopo alcune interazioni si ottiene quindi un istogramma che rappresenta l'andamento della **serie storica** dei punti delle storie da completare (figura 5).

Se la serie storica del totale punti **non scende** come previsto, il diagramma ci avverte immediatamente, dato che l'istogramma supera la retta del trend a zero.

Se questo avviene per una iterazione ma poi ritorna subito sotto, probabilmente non è il caso di allarmarsi. Se invece il superamento si dovesse manifestare per un periodo più lungo, allora è evidente che non si tratta di un caso isolato, ma anzi è probabile che la **Velocity** del team non sia compatibile con la data di termine che si era ipotizzata (figura 6).

Anche in questo caso il Burn-Down è utile perché consente di capire immediatamente quale potrebbe essere la nuova consegna: tracciando il nuovo trend a zero: si prendono i punteggi parziali degli ultimi due o tre sprint per avere una media

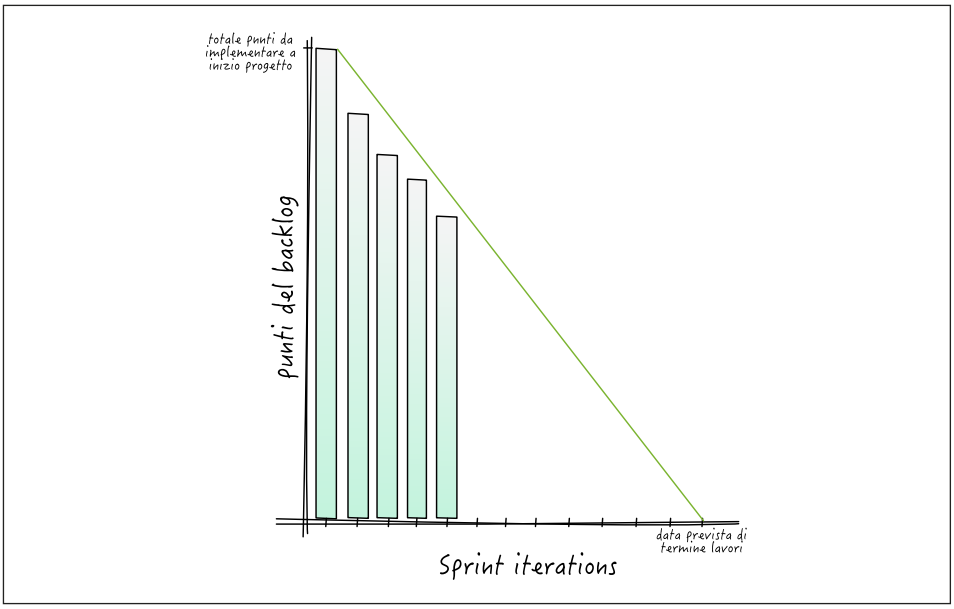


Figura 5. Dopo qualche iterazione si forma un istogramma che permette di individuare il trend dell'implementazione delle varie storie e consente quindi di capire se la data ipotizzata sarà rispettata.

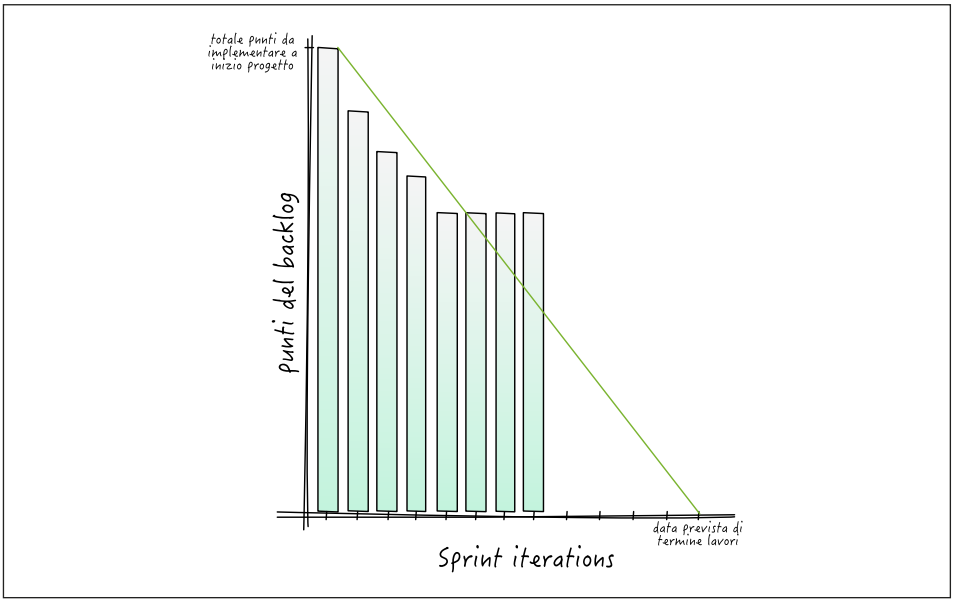


Figura 6. Il diagramma del Burn-Down visualizza in modo molto efficace se l'attuale trend non permette di rientrare nella data inizialmente stimata.

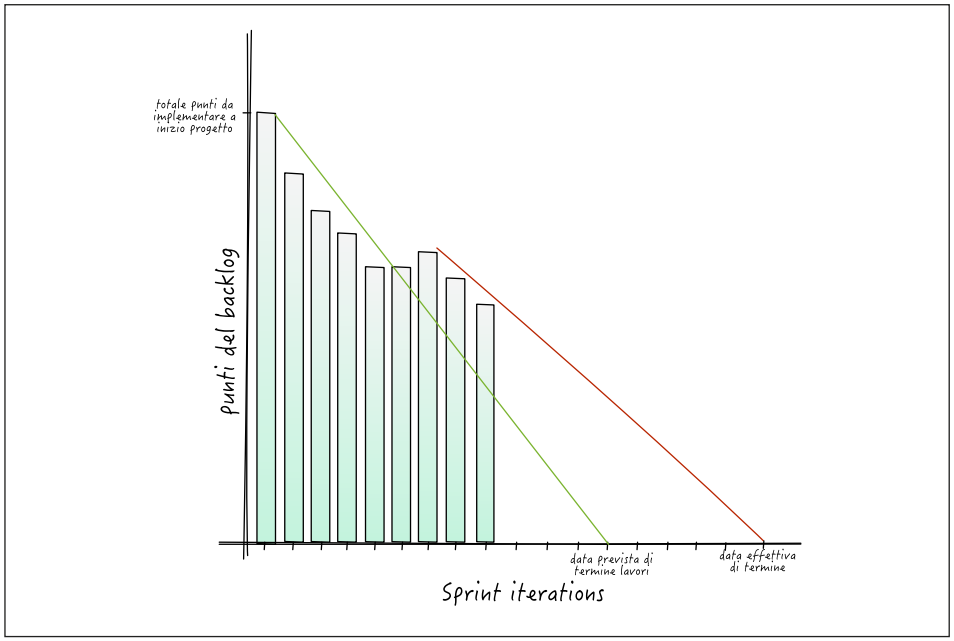


Figura 7. Se la velocità rallenta, si può ricalcolare la nuova data di consegna, semplicemente tracciando il nuovo trend a zero (in rosso) che interseca i punteggi degli ultimi due o tre sprint.

affidabile, si traccia la linea che li interseca e si ottiene la nuova data all'intersezione con le ascisse (figura 7).

L'affermazione di Scrum

Questa panoramica su Scrum mette in luce come esso non sia poi un'infrastruttura così complicata come qualcuno ha tentato di dipingerla. Il fatto che negli ultimi anni tale metodologia abbia riscontrato un'adozione progressivamente più ampia, consentendo la buona riuscita di molti progetti, ne testimonia la validità.

In Scrum esistono dei principi fondamentali o "valori" da tenere sempre presenti. Occorre conoscere svariati elementi chiave (eventi, ruoli e artefatti) che però non sono difficili da comprendere. Ci sono infine poche regole da rispettare, le quali non vanno intese come leggi calate dall'alto, quanto piuttosto come una serie di buone pratiche derivate dall'esperienza sul campo.

Nei suoi elementi costitutivi, quindi, Scrum non è una metodologia "difficile": quello che è impegnativo, casomai, è la sincera adesione ai suoi valori e l'appropriata applicazione delle pratiche nella quotidianità, senza farsi troppi sconti...

E di tutti questi aspetti operativi parleremo nel dettaglio nei capitoli che seguono.

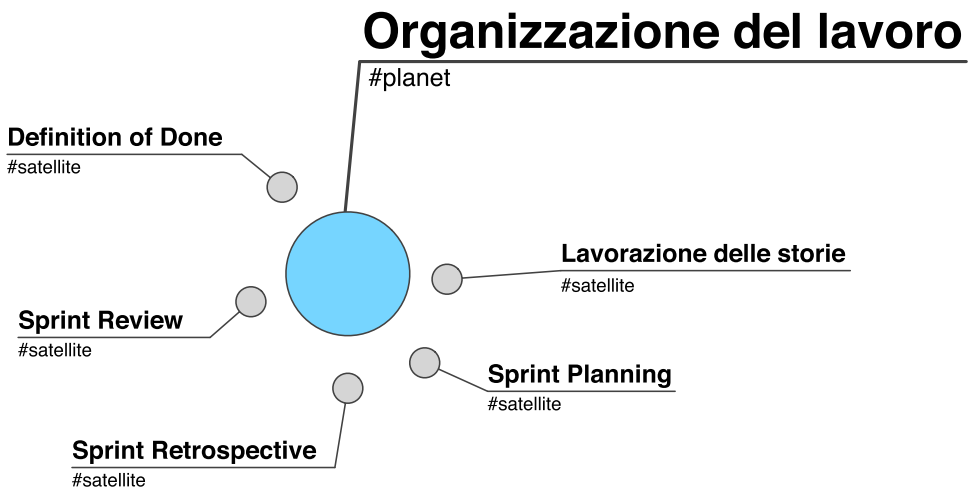
Riferimenti

[GS] K. Schwaber, Jeff Sutherland, *La Guida a Scrum™*
http://bit.ly/IZaZoI

[AC] Quali elementi individuano un coach agile?
http://blog.connexxo.com/2009/07/what-is-an-agile-coach-really.html/

[HL] Quali elementi costituiscono la host leadership?
http://hostleadership.ning.com/

Capitolo 2
Organizzazione del lavoro negli sprint



Strutturare le iterazioni

Sebbene al team Scrum sia lasciata ampia libertà di organizzare il proprio lavoro all'interno dello sprint, ogni iterazione deve seguire uno schema piuttosto preciso per quello che riguarda l'inizio e la fine. In questo capitolo analizzeremo le varie fasi e le attività collegate allo sprint.

La pianificazione dello sprint: lo Sprint Planning

A regime, durante la creazione del prodotto, Scrum dà per assodato che ci sia un **Product Backlog** contenente le varie funzionalità da implementare. Come avremo modo di vedere, il backlog è una struttura in costante **lavorazione** ed **evoluzione**: grazie al lavoro di **Refinement**, il team infatti fa emergere verso l'alto funzionalità sufficientemente dettagliate e di grana compatibile per essere messe in lavorazione in un'iterazione. Il **Refinement**, il cui scopo è quindi quello di preparare le storie utente necessarie per alimentare il lavoro del team, agisce sugli elementi che sono stati già inseriti all'interno del backlog. Non c'è una regola sul quando e su quanto tempo il gruppo debba dedicare a queste attività di trasformazione: il team si auto-organizza in modo che ci siano **sempre disponibili storie pronte** per la lavorazione all'interno dello sprint.

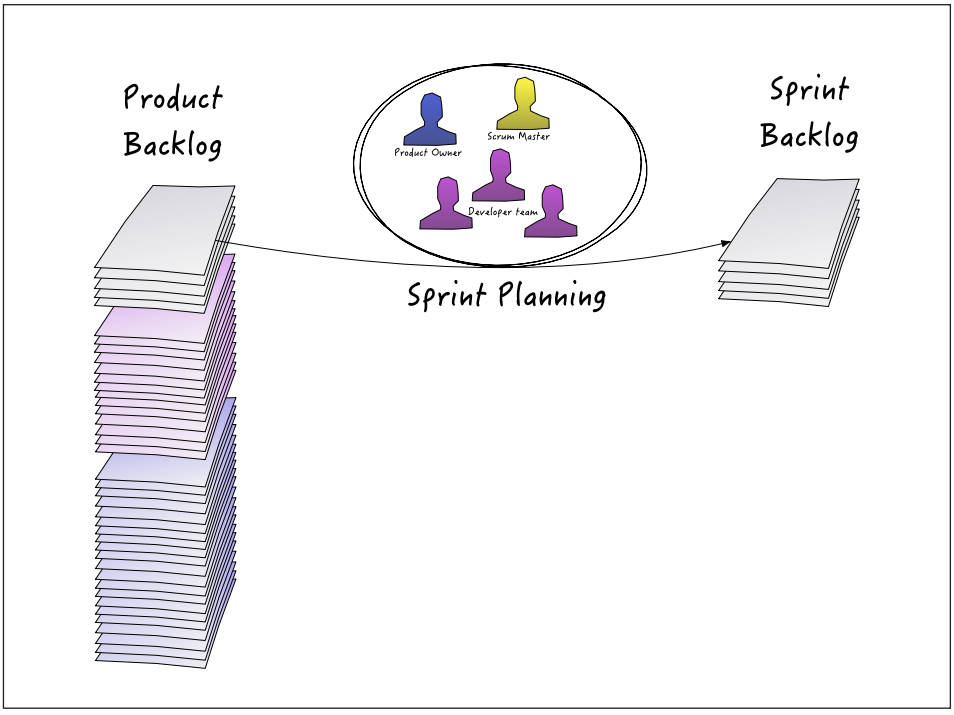


Figura 8. L'obiettivo dello Sprint Planning è di stabilire quante e quali storie del Product Backlog si potranno implementare nello sprint che sta per iniziare.

La composizione del **Product Backlog** è quindi frutto di un processo di raccolta dei requisiti, attività che rientra fra le responsabilità del PO, anche se non è detto che poi sia effettivamente lui a raccogliere i requisiti e a scrivere le storie. Normalmente questo processo di alimentazione prevede una prima **raccolta massiva** che permette di dar vita alla prima struttura grezza del backlog; successivamente al **Product Backlog** si possono aggiungere puntualmente nuovi elementi, qualora si evidenzi la necessità di inserire nuove funzionalità nel prodotto.

Dalla visione alla pianificazione

Questa fase di raccolta massiva degli elementi del Backlog non rientra fra gli obiettivi di Scrum, ma viene svolta tramite attività specifiche volte a trasformare una **Vision** di prodotto in una serie di funzionalità di alto livello. Fra i vari strumenti disponibili, molto utilizzati sono lo **User Story Workshop** e lo **Story Mapping**, di cui si parla dettagliatamente nella seconda parte di questo libro.

Ogni sprint inizia quindi con una attività di pianificazione detta **Sprint Planning**, in cui verrà stabilito quante e quali storie saranno implementate nella iterazione (figura 8).

Al planning partecipano:

- il **Product Owner** cui spetta la responsabilità di rispondere alle domande del team e di fornire la priorità sulle funzionalità da implementare
- il **Development Team** che deve valutare ogni singola storia in modo da comprenderne peso o complessità
- lo **ScrumMaster** il quale, se non fa parte del team di sviluppo, svolge in questo caso il ruolo di "cerimoniere".

La presenza del **Product Owner** è di vitale importanza per questa attività ed è per questo motivo che spesso egli nomina un vice che lo sostituisce qualora non possa presenziare alla riunione. Il Product Owner e il suo vice devono per questo motivo essere sempre allineati su ciò che deve essere fatto, sul contenuto e sull'ordinamento del **Product Backlog**, sui risultati dei vari **Refinement** del backlog e, più genericamente, sulla direzione da dare al progetto.

Scrum ufficialmente raccomanda che questa riunione duri al massimo **2 ore** per ogni **5 giorni** di sprint; quindi, se si è deciso che ogni iterazione duri due settimane (10 giorni di calendario lavorativo), la pianificazione durerà 4 ore.

Come avremo modo di vedere nel paragrafo successivo, è raccomandabile che il team investa tempo ed energie sulle attività di raffinamento in modo da **anticipare** il lavoro che si dovrebbe fare durante lo **Sprint Planning**: per questo motivo un team ben organizzato dovrebbe dedicare al massimo 1 ora al planning; e c'è chi suggerisce addirittura mezz'ora.

Le varie fasi del planning

Controllando quanto riportato all'interno della Scrum Guide, alla voce **Sprint Planning** si trova la seguente definizione:

"Lo Sprint Planning si compone di due parti, ognuna delle quali dura la metà della durata complessiva. Le due parti rispondono rispettivamente alle seguenti domande:
- Che cosa sarà rilasciato nell'incremento derivante dal prossimo Sprint?
- Come sarà fatto il lavoro necessario a garantire il raggiungimento dell'incremento?"

Lo scopo dello **Sprint Planning** è quindi di **impostare** il **lavoro** per il prossimo sprint: il **modo** con cui ottenere questo obiettivo è lasciato al team che può seguire varie strade.

Alle volte, specialmente quando il team ha raggiunto un discreto livello di maturità e di consapevolezza dei principi di agilità, il planning segue uno **schema libero**, in cui le persone del team si auto-organizzano e collaborano. Altre volte invece il gruppo sente la necessità di rispettare un **cerimoniale** piuttosto **rigido** in cui ogni persona lavora all'interno del proprio ruolo (**PO** e **Dev Team**): lo **Sprint Planning** in questo caso si svolge in modo molto più meccanico e schematico.

Il primo approccio, raccomandato nello **Scrum moderno**, è quello che permette una maggiore efficienza e una riduzione dei rischi. In questo caso, prima di arrivare al planning, il **PO** e il **Development Team** svolgono approfondite sessioni di **Refinement** nelle quali, oltre a lavorare sulla scomposizione degli elementi a grana più grossa, si eseguono indagini e approfondimenti per comprendere meglio il contenuto delle singole storie. In questi incontri si procede spesso anche a stimare l'effort necessario per l'implementazione, tramite una delle tecniche prevista nei progetti agili; molto utilizzata in tal senso la tecnica del **planning poker**, di cui si parlerà più avanti.

Sebbene sia compito e responsabilità del **PO** gestire il contenuto del **Product Backlog**, in questo caso spesso sono le persone del **Dev Team** stesso che si attivano per smontare gli elementi a grana grossa o per scrivere nuove storie: si tratta infatti di un'attività che il team di sviluppo avrebbe dovuto comunque svolgere più avanti. In questo modo, prima che si svolga lo **Sprint Planning**, il team riesce a sollevare i dubbi e le eventuali criticità delle storie.

Se invece il gruppo lavora in modo sommario e poco approfondito al raffinamento, tipicamente limitandosi a spezzare le storie a grana grossa, tutte le attività di indagine sono rimandate allo **Sprint Planning**. In questo caso quindi il **PO** presenta le storie descrivendone sia l'**obiettivo** che gli eventuali **criteri di accettazione**.

Il formato utilizzato per le storie può variare, anche se nella maggior parte dei casi si usa il formato introdotto anni fa in Connextra [CXT], basato sull'uso di cartellini a doppia faccia: sul fronte si trovano il **titolo** e la **descrizione**, mentre sul retro sono riportati i cosiddetti **Acceptance Criteria**, vale a dire i criteri in base ai quali si considera come completata la storia in questione. In ogni caso, di questi aspetti specifici delle storie utente parleremo nel dettaglio nel capitolo successivo in questa stessa parte.

Durante la presentazione di una storia — che sia fatta in fase di raffinamento o direttamente allo Sprint Planning — il team pone tutte le domande del caso in modo da avere il più chiaro possibile quale sarà il tipo di lavoro da fare. Lo scopo è quello di comprendere il tipo di lavoro da svolgere per poterne stimare l'**effort**.

Per questo motivo bisognerebbe limitare il desiderio di condurre la discussione nella disanima dei dettagli tecnici, come la struttura della GUI, la struttura del database sottostante o l'architettura di dettaglio del sistema; analogamente si dovrebbe ricordare che non è questo il momento di effettuare una dettagliata analisi funzionale. La discussione di ogni storia dovrebbe essere quindi una sorta di **impegno a rivedersi** in un momento successivo in cui si entrerà più nel dettaglio delle storie.

Anticipare o rimandare l'analisi delle storie?

Lo studio e l'approfondimento delle storie utente può essere quindi anticipato durante le sessioni di **Refinement**, oppure rimandato in ultima battuta al momento dello **Sprint Planning**.

Anticipare semplifica non poco lo **Sprint Planning** che quindi si riduce a una **rapida presentazione** da parte del **PO** delle storie che sono già state sufficientemente analizzate in precedenza; il lavoro del **Dev Team** quindi è di accettare le storie che sente di essere in grado di implementare. Anticipare è però anche un efficace modo, sebbene non l'unico, per **ridurre il rischio** sul progetto.

Viceversa **rimandare** studio e approfondimento delle storie utente allo **Sprint Planning** non è quindi una buona strategia: il rischio più evidente è quello di non riuscire a svolgere la riunione dello **Sprint Planning** all'interno del time box raccomandato dalla stessa Scrum Guide. Oltre a questo, aumenta considerevolmente la possibilità di scoprire dubbi o problemi quando è ormai troppo tardi per trovare una soluzione, obbligando **PO** e **Dev Team** a scegliere fra il **rimandare** la storia a uno Sprint successivo, così da avere tempo per studiare meglio il problema emerso, oppure mettere in lavorazione una **storia incompleta**, non pienamente definita o analizzata, con qualche problema pendente, perché non è possibile rimandare la sua implementazione. Infine, anche da un punto di vista dell'organizzazione del lavoro, questo approccio si dimostra non in linea con la definizione di **Sprint Planning** riportata nella Scrum Guide ufficiale:

"Il lavoro da svolgere nello Sprint è pianificato durante lo Sprint Planning Meeting. Questo piano è creato dal lavoro **collaborativo** dell'intero Team Scrum."

Rimandare la discussione delle storie allo **Sprint Planning** di fatto limita molto l'approccio collaborativo: il **PO** presenta le storie, il team di sviluppo ascolta e fa le domande, Il PO risponde. È una procedura in linea con le responsabilità dei vari ruoli, ma non dà molto il senso della collaborazione, come invece raccomandato anche da uno dei principi dell'Agile Manifesto: **Business e Dev Team lavorano insieme**. Svolgere il planning in questo modo è comunque un buon modo per collaborare, ma certamente non il migliore.

Perché aspettare lo Sprint Planning?

Anticipare lo studio delle storie ha quindi numerosi vantaggi, ma accade in svariati casi di imbattersi in team che si riducono a fare tali attività allo **Sprint Planning**; i

motivi possono essere i più disparati: dalla pigrizia o scarsa organizzazione del team per trovare il tempo necessario alla mancanza di un **PO** disponibile a una collaborazione frequente e approfondita. In svariati casi la "mancanza di tempo" è solo una scusa: il **Dev Team** nasconde così il desiderio di continuare a ricevere le indicazioni dal **PO**, invece di imparare a organizzare autonomamente il proprio lavoro.

Come termina il Planning

Lo scopo della **prima** parte dello Sprint Planning è rispondere alla domanda "Che cosa sarà rilasciato nell'incremento derivante dal prossimo Sprint?", ed è per questo che durante questa fase il team si focalizza sul **definire** il **contenuto** dello **Sprint Backlog**.

Indipendentemente dall'approccio adottato per il la pianificazione dello sprint, la valutazione sul numero delle storie che il **Dev Team** decide di accettare è di sua totale pertinenza. L'effort stimato per la lavorazione delle storie viene espresso in **punti**: di solito il gruppo confronta la somma dei punti delle storie che sta accettando con i punti che ha completato negli Sprint precedenti (la cosiddetta **Velocity**). Il confronto fra **punti fatti** in precedenza e **punti presi in carico** per lo sprint a venire fornisce spesso solo una indicazione di massima, dato che il gruppo di sviluppo può decidere in totale autonomia e libertà di fare più cose o di farne meno.

L'**ordine** degli elementi dello **Sprint Backlog** dovrebbe ricalcare quello del **Product Backlog**, in modo che il team sia stimolato a implementare per prime le storie che erano state inserite nelle prime posizioni del **Product Backlog**; ma questa non è una regola obbligatoria; al solito il team è libero, all'interno dello sprint, di organizzarsi come meglio crede.

Spesso il team utilizza una lavagna (o *board*) detta **Task Board** per la visualizzazione dello **stato** di **avanzamento** dei **lavori**; in questo caso, ogni storia che viene accettata durante il planning è poi appesa direttamene su tale lavagna in una colonna che andrà poi a formare lo **Sprint Backlog** (figura 14). In genere la forma e il funzionamento di tale lavagna ricordano molto quella utilizzata in **Kanban**; ma di Kanban parleremo diffusamente nella parte 4 di questo libro.

La **seconda** fase dello **Sprint Planning** serve invece per rispondere alla domanda "Come sarà fatto il lavoro necessario a garantire il raggiungimento dell'incremento?".

Sebbene la guida ufficiale non fornisca alcuna indicazione in tal senso, spesso il team di sviluppo si organizza per smontare le singole storie in attività (dette anche *task*) per avere un'idea più precisa del reale lavoro da svolgere nello sprint che sta per iniziare. La **suddivisione** delle **storie** in **attività** spesso viene suggerita come tecnica di scomposizione quando le storie portate nello **Sprint Backlog** sono ancora troppo grandi rispetto a come dovrebbero essere.

Qualora il team sia composto da persone con competenze non completamente crossfunzionali, la scomposizione delle storie potrebbe essere effettuata secondo le mansioni delle persone, ossia analisi, design, implementazione e test. È certamente questa una soluzione subottimale, sia dal punto di vista dell'efficienza poiché introduce

dipendenze fra le persone, facilita lo sbilanciamento del carico di lavoro, non fa crescere le persone, sia e soprattutto dal punto di vista del processo, visto che in questo modo si introduce un "mini-waterfall" all'interno dello Sprint. Ciò nonostante, quando il team sia composto da persone con competenze verticali (team **non crossfunzionale**), questo tipo di scomposizione permette di procedere all'implementazione delle storie e, qualora l'organizzazione lo consideri un obiettivo, di far evolvere il team di sviluppo verso una reale interscambiabilità delle persone e delle competenze.

Interessante notare che nelle prime versioni di Scrum si dava la possibilità al team di organizzare il progetto in **iterazioni** molto **lunghe**, anche di qualche **mese**: in quel contesto la scomposizione dava luogo a molti task che erano un ottimo modo per misurare lo stato di avanzamento dello sprint: solo i task completati danno l'idea precisa dello stato di avanzamento.

Gestione del backlog e negoziazione delle attività da svolgere nello sprint

Durante lo **Sprint Planning** talvolta possono insorgere delle tensioni quando il desiderio e le aspettative del **PO** non sono in linea con la visione del **Dev Team**. I due ruoli spesso hanno un diverso approccio nel valutare il totale delle cose da fare: il primo, tipicamente, ragiona in **numero di storie** messe in lavorazione, mentre il team di sviluppo basa le proprie valutazioni ragionando in termini di **effort necessario** per implementare le singole storie, espresso per esempio con il sistema dei **punti lavorabili** in uno sprint. Queste due grandezze, pur facendo riferimento allo stesso concetto di base (**quantità di lavoro da svolgere**), spesso non sono in accordo.

Scrum gestisce questa dicotomia da un lato imponendo prima di tutto una chiara separazione dei compiti: il **Product Owner** decide il **contenuto** e l'**ordine** dei vari elementi del **Product Backlog**, il **Dev Team** stabilisce **quanti** di questi elementi potranno essere presi **in lavorazione** all'interno dello sprint. Oltre a questo, la metodologia prevede e incentiva la discussione in modo da raggiungere un compromesso utile per la buona riuscita del progetto.

Si immagini per esempio il caso in cui il **Product Owner**, durante il planning, si aspetta che il team accetti le prossime 5 storie del **Product Backlog**. Durante il planning invece il gruppo di sviluppo potrebbe decidere di rifiutare di lavorare la quinta storia perché ritiene di non riuscire a portarla a completamento all'interno dell'iterazione.

Nell'esempio la quinta storia dovrebbe essere quindi inserita in uno degli sprint successivi, cosa che potrebbe essere in contrasto con le aspettative e la pianificazione del **Product Owner**: egli potrebbe, per esempio, aver necessità di mostrare al cliente, al più presto, tutte e cinque le storie oppure solo la quinta.

Il **PO** non può imporre l'accettazione della quinta storia ma può apportare tutte le modifiche al **Product Backlog** per permettere che anche questa storia sia messa in lavorazione: per esempio potrebbe invertire l'ordine delle storie in modo che la quinta finisca in testa all'elenco delle cose da fare; in tal caso rimarrebbe fuori dallo sprint una delle altre quattro. Un'altra opzione potrebbe essere quella di "alleggerire" tutte le storie

presentate in modo da ridurre l'effort necessario per implementarle e quindi permettere che nello sprint siano accettate cinque storie anziché quattro: quello che toglie da una storia potrebbe poi essere inserito in un'altra che verrebbe eseguita più avanti nel progetto; in tal caso si parla di "storia di integrazione".

Negoziare con la discussione

La discussione che avviene durante il **Planning** (o meglio durante il **Refinement**) è quindi un momento estremamente importante perché permette di condividere informazioni e perché stimola la **crescita professionale** delle persone: il PO migliora nell'attività di **confezionamento delle storie**, il team di sviluppo, pur focalizzando la sua attenzione sugli aspetti implementativi, impara sempre più a fornire indicazioni sul come **aiutare il PO** nelle sue scelte di business. Il formato delle storie, fatto di un corpo ma anche di una lista di **criteri** di **accettazione** modificabili singolarmente, agevola questa discussione. Per chi fosse interessato ad approfondire questi aspetti, si consiglia la lettura del testo *Fifty quick ideas to improve your user stories* [IS].

Ogni volta che il **Product Owner** presenta una storia, il team di sviluppo ne valuta la difficoltà cercando di ricavare tutte le informazioni possibili e di esprimere quindi una stima in punti. Per ogni nuova storia aggiunta, il team aggiorna il punteggio totale dei punti in lavorazione sommando i punti delle storie contenute nello sprint backlog.

La Velocity

Per decidere quante storie accettare in lavorazione nello sprint, il team si aiuta confrontando la somma dei punti fatti in ognuno degli sprint precedenti, la cosidetta **Velocity del team**, con la somma dei punti della storie che si intendono accettare. Importante tener presente che la **Velocity** può essere presa come **indicatore** della capacità di lavoro del gruppo, ma spesso la decisione finale è frutto anche di altre considerazioni, come una maggior confidenza dei propri mezzi, la crescita o diminuzione della complessità del progetto e così via.

Interessante notare come il valore **atteso** della **Velocity**, vale a dire quanto il team ipotizza di produrre a inizio sprint, ha un andamento piuttosto altalenante durante i primi sprint, ma finisce per stabilizzarsi dopo alcune iterazioni; il grafico riportato in figura 9 esprime esattamente questo fenomeno. Il processo iterativo e in particolare l'analisi **retrospettiva** di fine sprint permettono al team di prendere realisticamente coscienza delle proprie capacità produttive. La **Velocity**, o meglio la sua stima, diventa quindi una grandezza piuttosto affidabile del numero di storie lavorabili all'interno di uno sprint e quindi anche del completamento dell'intero prodotto.

NOTA: Accanto al termine **Velocity** si trova spesso quello di **Capacity**; le due definizioni sono spesso utilizzate l'una al posto dell'altra con lo stesso significato, anche se alcuni danno alle due parole una leggera differenza: per **Velocity** intendono infatti la somma dei punti delle storie lavorate con successo, ossia i punti realmente prodotti, mentre con **Capacity** si intende

la previsione di punti che il team potrà lavorare per il prossimo sprint. In questo caso la Capacity viene calcolata moltiplicando i punti/persona fatti nell'ultimo sprint (valore ottenuto dividendo la Velocity fatta per il numero di persone che compongono il team di sviluppo) per il numero di giorni-uomo disponibili nello sprint che sta per iniziare, al netto degli eventuali giorni di ferie, permessi, altri impegni, part-time su altri progetti.

Stimare le storie: il planning poker

Per valutare il tempo necessario a completare una storia in Scrum si usa valutarne il **peso** (ossia l'effort necessario per il completamento) espresso tramite un **punteggio numerico**.

Figura 9. La velocity del team rappresenta la quantità di cose che il team riesce a implementare in uno sprint. Si misura in punti ed è valutata in modo empirico a posteriori. Il suo valore viene sempre aggiornato, sprint dopo sprint, in modo da avere sempre un valore medio approssimativo affidabile. È un numero che può cambiare nel tempo e che si stabilizza su asintoti più o meno costanti.

NOTA: La complessità o la difficoltà del lavoro da svolgere per implementare una storia non permettono di ricavarne il tempo necessario, dato che per esempio una attività semplice ma molto ripetitiva potrebbe richiedere più tempo di una complessa ma puntuale. Si pensi per esempio alla messa a punto del template di stampa di una fattura: si tratta in genere di una attività estremamente semplice (mettere a punto le coordinate in mm delle varie linee di stampa), ma altrettanto dispendiosa in termini di tempo.

Per ottenere il peso di ogni storia, si possono seguire varie tecniche anche se probabilmente la più usata è quella del **planning poker** [PP]. Il planning poker si basa sulla tecnica di stima **Delphi** [DE] inventata negli anni Quaranta, ripresa nel 1968 e infine affinata nel 2002 da James Grenning; è stata infine resa popolare da Mike Cohn nel libro *Agile Estimating and Planning* [AEP].

Si tratta di un metodo di lavoro **collaborativo** molto semplice che è organizzato in forma di gioco. Lo scopo è cercare di far emergere il punto di vista di tutti, evitando di intavolare lunghe e inutili discussioni, sia per la scarsità di informazioni a disposizione sia perché in tal modo si finirebbe di dar vita a una qualche forma di waterfall.

Per questo, durante il planning, ci si limita a fornire una valutazione di massima, rimandando a quando la storia verrà implementata gli approfondimenti necessari.

Figura 10. Con il planning poker, in un ambito rilassato e divertente, tramite carte da gioco, biglietti, le dita delle mani o una app sullo smartphone, si arriva a votare, vale a dire "pesare", la complessità in punti di ogni storia.

Le regole del gioco

Al momento del voto lo ScrumMaster dice: "OK ragazzi procediamo al voto: 1, 2, 3... votate!". In questo momento, e non prima, per non influenzarsi a vicenda, tutti i membri del team esprimono la propria valutazione.

Ogni partecipante, per formulare la propria valutazione sulla storia, le assegna un punteggio utilizzando un valore preso da una scala concordata con gli altri partecipanti. Per votare si possono usare delle **carte da gioco** opportunamente stampate (se ne trovano in commercio moltissimi tipi), dei foglietti di carta con su scritti i numeri, le mani, oppure, nella forma più moderna, una delle molte **app** utilizzabili sul proprio smartphone o una web application utilizzabile tramite un comune browser: soluzione utile, quest'ultima, per esempio nel caso di team distribuiti.

Il voto viene espresso utilizzando punteggi presi da una scala discreta non lineare (la scelta non è casuale, vedi oltre) e per questo motivo spesso si attinge alla prima parte della serie di Fibonacci (0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...) alla quale si aggiungono alcuni valori particolari come ½ per indicare attività minimali, **200** o **400** (per indicare che la storia è ancora molto grande, di fatto non è lavorabile dato che si tratta di una "epica"), **infinito** o **?** ("non ho informazioni per pensare a un numero"), il simbolo **π** (*pee*, "ho bisogno di interrompere un momento per fare pipì") che a volte è sostituito dal simbolo di una **tazzina** ("ho bisogno di una pausa per un caffè").

Capita piuttosto spesso che alla prima votazione non si raggiunga l'unanimità sul peso da attribuire alla singola storia: in questo caso **si svolge una rapida discussione** a cui partecipano due persone prese fra coloro che hanno espresso i **voti più lontani** fra loro. Si immagini per esempio che il risultato della votazione siano i seguenti punteggi 2, 3, 5, 5, 3, 21: in questo caso si assume che 3 e 5 siano valori in un qualche modo paragonabili, per cui si cerca di capire perché sono stati votati i numeri 2 e 21. Quindi chi ha votato 2 esprime le sue ragioni e lo stesso fa chi ha votato 21: per evitare che la discussione si protragga indeterminatamente, la regola è che solamente queste due persone hanno diritto di parola e che possano argomentare la loro scelta una sola volta con al massimo un diritto di replica. Dopo la spiegazione dei voti divergenti, si vota nuovamente e si prende come risultato finale un valore medio o quello approssimato per eccesso (riportato sul primo numero di Fibonacci approssimando per eccesso, ossia, se la media è 4, si prende 5).

Lo scopo di questo regolamento è quello di evitare l'insorgere di discussioni lunghe e infruttuose: come accennato in precedenza, viste le poche informazioni a disposizione in questa fase, oltre un rapido confronto, proseguire ulteriormente non porterebbe a miglioramenti significativi.

Il meccanismo di stima non segue alcuna regola matematica ma si basa su un approccio estremamente empirico e pragmatico: il team "ipotizza" un peso per ogni storia, e il processo ripetitivo (verrà eseguito ad ogni inizio sprint) e la retro-valutazione a fine sprint, garantiscono risultati migliori di ogni altra forma di stima basata su formule o equazioni. Si tratta di un sistema di valutazione adattivo: spesso infatti i punteggi

espressi sulle storie, nonché il calcolo della capacity di sprint sono affette da un errore piuttosto elevato alle prime iterazioni, ma finiscono per essere estremamente precise con il passare del tempo.

Stimare comparativamente

Un aspetto molto importante da tener presente è che questo sistema di stima è efficace proprio perché utilizza alcuni meccanismi tipici del nostro cervello. Uno di questi è la **valutazione comparativa**.

Se ci venisse chiesto per esempio di stimare il numero di chicchi di riso contenuti in un vaso di vetro, probabilmente forniremo un numero molto poco affidabile; anche nel caso riuscissimo a indovinarlo, non potremmo dire che la risposta sia affidabile, vista l'impossibilità di ripetere la performance con altri contenitori. Se invece ci venisse chiesto, di fronte a tre contenitori differenti, quale sia quello che contiene il maggiore quantitativo di chicchi di riso, probabilmente potremmo rispondere in modo estremamente affidabile.

Seguendo questo approccio si potrebbero quindi comparare i tre contenitori: A contiene meno riso di B che a sua volta ne contiene meno di C. Questa tipo di stima risulta semplice per il nostro cervello perché, oltre ad utilizzare un processo comparativo, utilizza una valutazione discreta: ci viene chiesto infatti di effettuare un confronto fra tre contenitori e non di stimare il volume di riso che passa per esempio su un nastro trasportatore.

Per un team Scrum è quindi più semplice stimare le storie secondo un approccio comparativo e utilizzando una scala discreta piuttosto che ipotizzare il tempo necessario per svolgere un determinato lavoro: molteplici sono i fattori in gioco, non ultimi la capacità del team, ossia di chi esegue quel lavoro.

Per permettere questa **stima comparativa discreta** la prima operazione da svolgere è la definizione della scala, cosa che può essere eseguita in vari modi. Uno di questi consiste nell'identificazione della **storia campione** che verrà poi usata come metro di paragone per tutte le altre. La storia campione viene scelta provando a identificarne una che richieda un tempo di lavorazione ben definito e piccolo, per esempio una giornata (figura 11). Specialmente se il team è inesperto di Scrum, punti e planning poker, in questa fase l'associazione punti-tempo è necessaria per consentire al gruppo di innescare il processo di valutazione; ma con il tempo queste due grandezze torneranno a essere slegate fra loro.

Successivamente si dispongono le carte con i punteggi su un tavolo molto grande — oppure si attaccano a una parete o si allineano sul pavimento — e si mette la storia campione vicino al punteggio medio prescelto: la scala è relativa per cui non ha molta importanza il valore scelto per tale storia. Infine si confronta ogni altra storia con la storia campione, provando a valutare se è più grande o più piccola: se è più grande, si prova a capire "Quanto più grande? Il doppio? Il triplo? Dieci volte?" e in base alle risposte si colloca la storia più o meno vicina alla storia campione (figura 12).

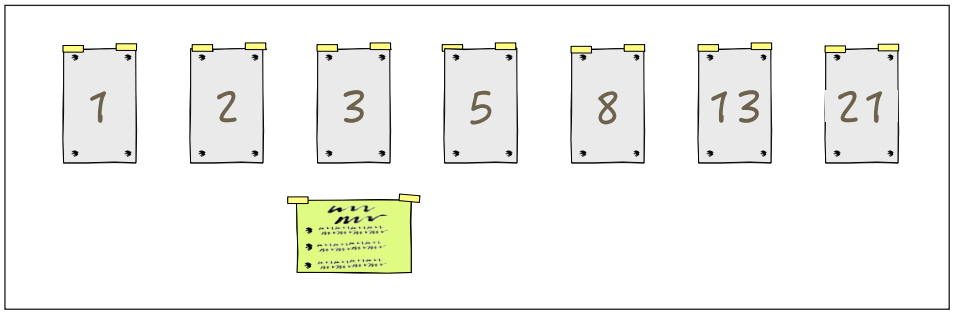


Figura 11. La prima fase del processo di stima delle storie è identificare una storia di campione, per la quale il team è confidente che il tempo di lavorazione sia dell'ordine di una giornata. Dopo aver disposto le carte con i punteggi su un piano o su una parete, si associa la storia campione alla carta con un punteggio piccolo, per esempio 3 punti.

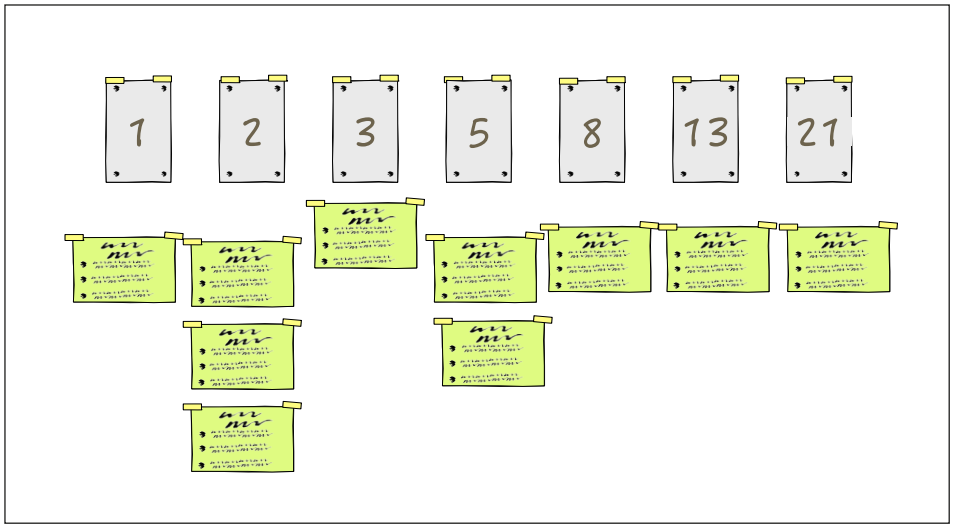


Figura 12. Successivamente, in base a un processo di confronto, si posizionano le altre storie, cercando di ipotizzare in che rapporto stanno (in termini di effort) rispetto alla storia campione.

In questa fase di setup è accettabile lavorare in maniera intuitiva e approssimativa, visto che le poche informazioni in possesso all'inizio di un progetto non consentirebbero di ottenere risultati più precisi.

Gli eventuali errori derivanti da una scelta sbagliata della storia campione, così come l'inesatto posizionamento delle storie rispetto alla scala, sono ininfluenti; di fatto si bilanciano dopo poche iterazioni tanto da rendere inutile in questa fase ogni tentativo di

raffinare le scelte di pesi e scale. Le tre grandezze (stime in **pesi**, taratura della **scala** e **Velocity** del team) sono collegate fra loro a formare un **sistema dinamico** e **auto-adattivo**.

Per esempio, se il team si accorge che l'effort necessario per realizzare la storia campione è minore del previsto, e che quindi la storia campione è stata valutata troppi punti, potrà correggere prendendo in lavorazione un numero maggiore di storie nello sprint. I punteggi restano costanti, il rapporto di proporzionalità pure ma aumentano i punti della Velocity.

Cosa rappresentano i punti delle storie

I punti usati per le stime sono, come visto, presi da serie numeriche che non sono collegate ad alcuna unità di misura: i punti rappresentano una **grandezza adimensionale**. Questa scelta permette di concentrarsi sul peso di una storia e non sul tempo di lavorazione, che dipende da molti fattori, e come abbiamo visto permette di adattare valutazioni e di aggiustare il tiro dopo poche iterazioni.

Il **tempo** di **lavorazione** è invece una grandezza che è funzione della complessità e dei fattori ambientali e per due storie simili può cambiare con il tempo.

Il punteggio indica l'**effort** — non il tempo — necessario per implementare una determinata storia: quanto lavoro si riesce a svolgere per ogni ora dipende dall'efficienza con cui si può lavorare, tenendo quindi conto delle interruzioni, del livello di concentrazione delle persone, del dover svolgere altre attività esterne al progetto, come riunioni o attività amministrative.

Stimare in ore vuol dire scegliere un valore che dipende non solo dalla dimensione reale del lavoro ma anche da come il team "funziona" all'interno dell'organizzazione. Spesso un buon team Scrum migliora le proprie performance e quindi migliora la propria velocità, anche se il lavoro da fare per unità di punto rimane lo stesso. Stimando in **story point**, la ri-stima del tempo avviene automaticamente essendo ricalcolata in base alla **Velocity**. Se la stima invece è in ore, è necessario ogni volta adattare in base all'efficienza.

Ciò nonostante, c'è sempre il desiderio, prima o poi, di tradurre i punti in unità temporali: la domanda tipicamente è "OK, mi è chiaro che 8 punti rappresentano una grandezza adimensionale; ma quanto tempo ci vorrà a fare questa storia? Altrimenti come posso fare una stima sulla data di consegna da dare al cliente? Qual è la formula che traduce i punti in ore?". La risposta è che **non** si possono **tradurre direttamente** punti in ore se non valutando la velocità del team, che a sua volta è funzione di vari aspetti.

Fra i vari esempi e metafore che si possono utilizzare, uno che mi piace particolarmente è quello dell'ascesa in bicicletta di una salita di montagna. Ci si potrebbe, infatti, interrogare su quale sia il tempo necessario per percorrere una qualsiasi salita di seconda categoria di una tappa di montagna del Giro d'Italia. Appare ovvio che la risposta che tutti daremmo è che dipende da chi fa la salita, oltre che da altri fattori: per esempio, il tempo atmosferico, l'andatura impressa per arrivare ai piedi della salita, il tipo di bicicletta, la presenza o meno di compagni di squadra che aiutano dettando il ritmo e così via. Il tempo impiegato da un semplice appassionato sarà certamente maggiore di quello di un

dilettante ben allenato, che a sua volta impiegherà più tempo di un professionista iscritto al Giro d'Italia in lotta per la classifica del Gran Premio della Montagna.

I punti di una storia sono quindi paragonabili alla categoria di una salita (*hors catégorie*, prima, seconda, terza): sono misure adimensionali che esprimono la complessità o la difficoltà della singola storia in modo relativo alle altre. Analogamente, il tempo di lavorazione dipende quindi da una serie di fattori come la capacità del team di sviluppo, la capacità del Product Owner nel confezionare o la chiarezza e stabilità dei requisiti.

Per questo motivo, per esempio, le **performance di due team** non sono paragonabili, così come non sono paragonabili le Velocity che uno stesso team esprime in due progetti differenti: cambia il contesto e cambia l'esperienza del team stesso. A volte invece, purtroppo, la velocità di un team è vista come misurazione delle performance di un team.

La lavorazione delle storie durante lo sprint

Dopo aver concluso l'attività di planning, il team inizia quindi lo sprint analizzando l'elenco delle storie accettate e che compongono lo **Sprint Backlog**. Una delle prime cose da fare, se non già fatta durante la fase di verifica del planning, è quella di scomporre le storie in sotto-attività o **task**: per fare questo, il team, se ha scelto di usare una **task board** fisica, spesso si aiuta apponendo sulla storia utente dei bigliettini colorati, ognuno dei quali rappresenta una specifica attività (figura 13).

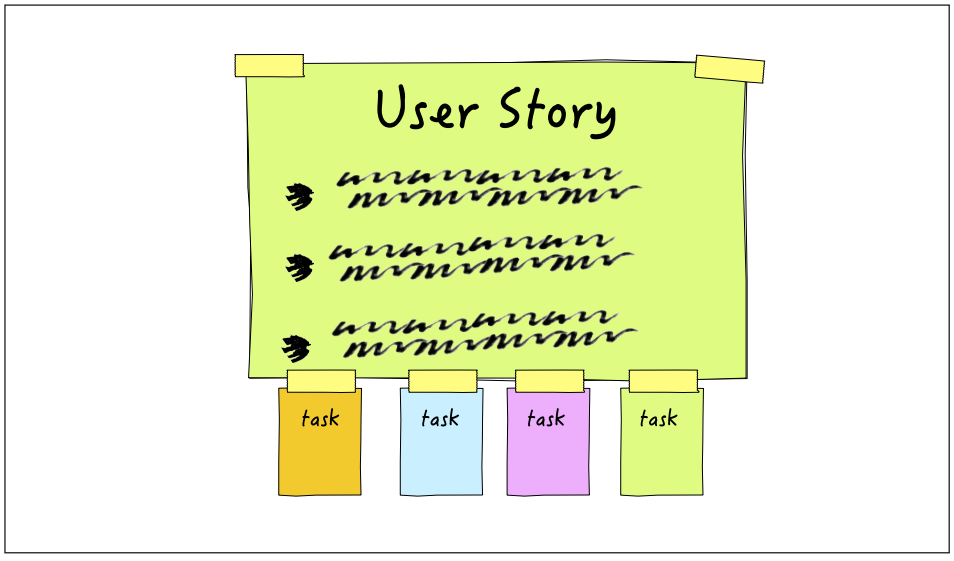


Figura 13. Scomposizione di una storia in task. Per ogni attività si attacca al cartellino della storia utente un bigliettino più piccolo, con un codice colore concordato.

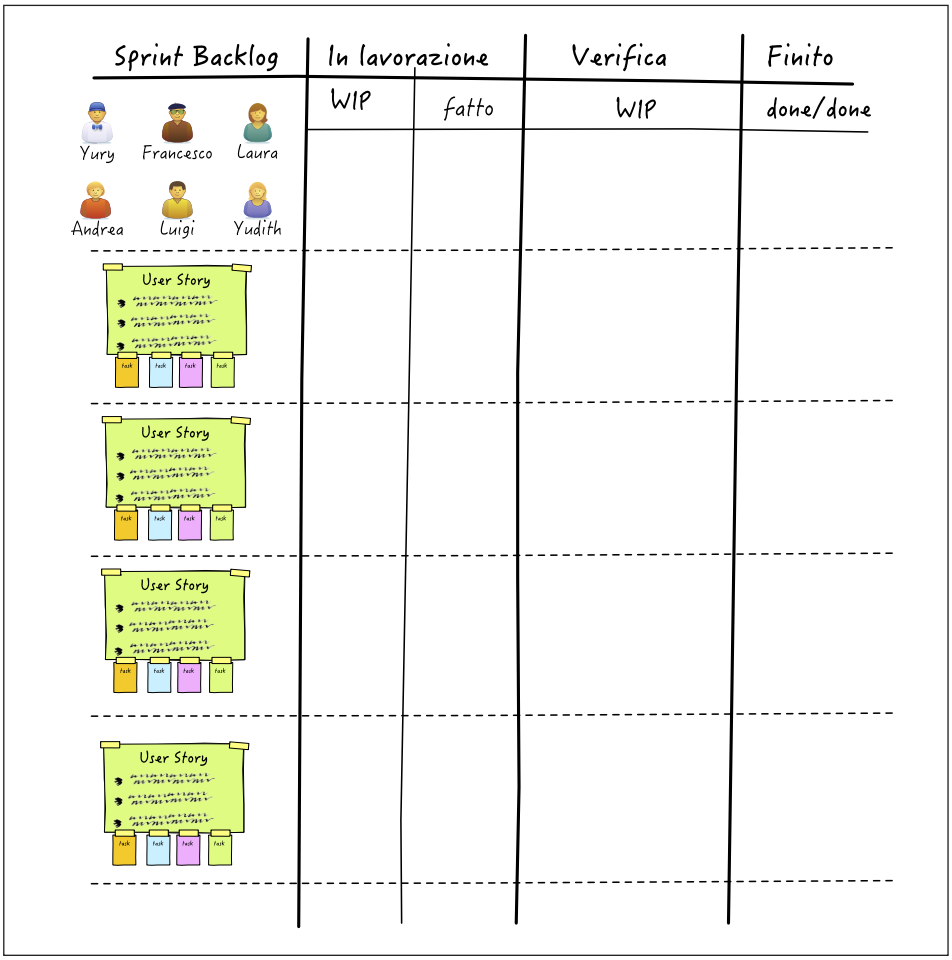


Figura 14. Le storie sono organizzate su una Kanban board che permette di seguire la lavorazione sia delle sotto-attività che delle storie nel loro complesso.

Se il team è composto da persone con competenze specifiche (analista, DBA, architetto, programmatore), la scomposizione della storia in genere segue questa organizzazione, per esempio creando task di analisi, di sviluppo, di documentazione, di test. Se il gruppo è invece pienamente cross-funzionale, è possibile, e certamente preferibile, effettuare la scomposizione delle storie per funzionalità verticali in modo che ogni task completato dia vita a qualcosa di testabile su tutta la filiera (p.e. dalla GUI al DB).

Il team si dota in questa fase di qualche strumento per monitorare l'avanzamento delle storie all'interno dello sprint: molto utilizzato è lo **Sprint Burn-Down Chart**, tool analogo al Product Burn-Down Chart di cui si è parlato in precedenza ma che in questo caso serve per monitorare la lavorazione dei task.

Oltre allo Sprint Burn-Down, un altro strumento molto utilizzato è la **Task Board** (figura 14), versione semplificata di una **Kanban Board** (per la quale si rimanda ai capitoli della parte 4 di questo libro). In questo caso si utilizza solo una parte delle indicazioni della metodologia inventata in Toyota anche se restano validi alcuni principi fondamentali: limitare il numero delle attività contemporaneamente in lavorazione, prediligere l'approccio *pull* e non *push*, dare priorità al completamento delle storie già iniziate piuttosto che a iniziarne di nuove. Son tutti concetti che approfondiremo nella parte 4 del libro.

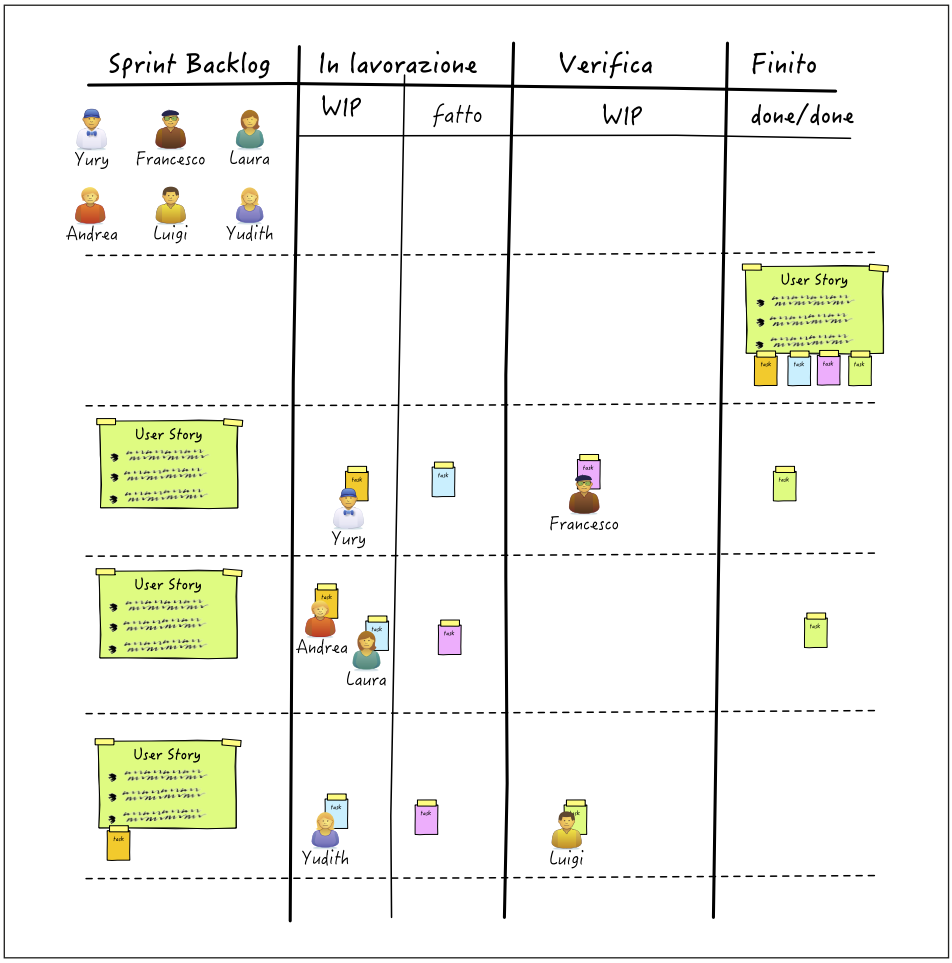


Figura 15. Grazie all'organizzazione del processo in fasi e sotto-fasi (WIP/done) ogni task può essere parcheggiato, appena si è terminata una fase della lavorazione, e analogamente prelevato per la lavorazione successiva, non appena si libera una persona in grado di eseguire la lavorazione successiva.

Tipicamente la board prevede una **colonna** per le cose **da fare** che rappresentano di fatto lo **Sprint Backlog**, mentre le **colonne successive** rappresentano i vari step del **processo** di lavorazione. Le tecniche di progettazione e le implicazioni delle varie configurazioni sono argomenti trattati in dettaglio nella parte di questo libro dedicata a Kanban.

Quando il lavoro dello sprint comincia, le persone del team prendono in carico le varie attività: la scelta su come procedere nella assegnazione dei task è di totale responsabilità dei membri del team, i quali possono decidere se dedicarsi all'intero ciclo di lavorazione di tutti i task di una storia o se invece seguirne solo alcuni dall'inizio alla fine. Sfruttando la board, man mano che la lavorazione sui task procede, i relativi cartellini sono spostati sulle varie colonne della board.

Per procedere nel lavoro di implementazione, ogni membro del team si attiva per reperire tutte le informazioni necessarie per portare a termine il proprio lavoro: potrà fare domande al **PO**, a un eventuale utente, a un analista o a chiunque possa rispondere ai suoi dubbi.

Come stabilire quando le attività sono completate: la Definition of Done

Man mano che si procede con la lavorazione della storia, i cartellini corrispondenti alle attività attraversano le varie colonne sulla board. Oltre alla lavorazione di tutti i cartellini, per poter considerare una storia completamente implementata, essa deve soddisfare i criteri di valutazione specifici per quella storia, la cosiddetta **Acceptance Criteria List** (**ACL**), oltre a una serie di criteri validi per tutte le storie e che sono inseriti nella **Definition of Done** (**DoD**).

La **ACL** è una lista che contiene i **requisiti funzionali** della storia stessa e per questo la sua compilazione e modifica è responsabilità del **Product Owner**.

La **DoD** invece contiene l'elenco dei **criteri** utilizzati per la validazione di **tutte storie** e per questo tipicamente contiene i **requisiti non funzionali**, quali, a esempio, il superamento dei test unitari o di integrazione, i check sulle performance o il completamento della documentazione utente allegata.

Non è detto che tutti i requisiti non funzionali finiscano all'interno della **DoD**: a volte infatti può essere necessario dover imporre un requisito non funzionale specificamente per una singola storia. Si pensi a quando si rende necessario utilizzare un protocollo di comunicazione cifrato solamente per la funzionalità di login. In questo caso si potrebbe inserire questo requisito di protezione del canale di comunicazione fra i criteri di accettazione specifici per quella storia.

L'operazione inversa invece — imporre un requisito funzionale a tutte le storie inserendolo nella **DoD** — per ovvi motivi non si verifica mai e per questo i requisiti funzionali sono inseriti nella lista degli **Acceptance Criteria**.

Maggiore è il numero e la complessità dei requisiti inseriti nella DoD, maggiore la mole di lavoro da svolgere per completare ogni storia. Per questo il contenuto di questa lista deve essere scelto con grande attenzione e giudizio: aggiungere o togliere un

criterio potrebbe spostare di molto le stime o al contempo abbassare o alzare la bontà del prodotto finito. In alcuni casi, l'organizzazione o il team di sviluppo potrebbero non essere pronti per soddisfare quanto richiesto dalla **DoD**: si pensi alla richiesta di svolgere i test di integrazione in una azienda che non abbia ancora definito alcuna policy e infrastruttura di **continuous integration**.

La **DoD** è quindi una lista di requisiti da rispettare rigidamente, ma non è detto che la lista stessa debba essere gestita in modo altrettanto rigido. A volte si decide per esempio di rilassare qualcuna delle restrizioni imposte dalla **DoD** in modo da rendere la velocità di lavorazione compatibile con le tempistiche di progetto. Altre volte, invece, nell'ambito di un processo di miglioramento continuo, si può decidere di aggiungere con il tempo qualche requisito quando l'organizzazione sia pronta per gestirlo.

Proprio per questo motivo, alcuni introducono il concetto di **Undone List**, per indicare tutti quei requisiti che l'organizzazione vorrebbe inserire nella **Definition of Done** ma che non è ancora pronta a implementare. Con il tempo gli elementi della Undone potranno essere trasferiti nella **Definition of Done** aumentando il livello di verifiche sulle attività.

È bene tener presente che ogni alterazione della **DoD** ha delle implicazioni non banali sia sulle storie già completate che su quelle da realizzare; nel primo caso l'aggiunta di un requisito non funzionale per esempio implica di dover apportare le necessarie modifiche alle storie che si riteneva fossero già terminate; nel secondo invece si potrebbe dar vita a una serie di componenti del prodotto costruiti su standard qualitativi più bassi.

Valutazione del prodotto finito: Sprint Review

Come si è già detto, lo Scrum moderno consiglia di verificare le varie storie non **appena** queste sono **pronte** (completate e verificate) senza attendere la fine dello sprint. L'obiettivo è quello di anticipare la scoperta di errori di interpretazione dei requisiti o di implementazione, in modo poter intervenire prima della fine dell'iterazione.

Ciò nonostante, al termine dello sprint il team esegue una delle attività più importanti, la **Sprint Review**, ossia la verifica di tutte le storie prodotte; questa attività segue un cerimoniale piuttosto semplice che, nel caso delle storie già controllate e approvate durante lo sprint, si riduce al minimo.

Il processo di verifica di una storia consiste prima di tutto nel valutare il rispetto degli **Acceptance Criteria** e nell'assicurarsi che i requisiti della **DoD** siano soddisfatti. Successivamente, se possibile, si esegue una vera e propria "demo" alla quale partecipano tutti membri del **team di sviluppo**, lo **ScrumMaster** e ovviamente il **Product Owner**. Durante questa dimostrazione, il team di sviluppo mostra al **Product Owner** ogni singola storia completata: se la storia lo permette, il modo migliore di effettuare questa demo è mostrare dal vivo la funzionalità implementata. Il **PO**, tramite la verifica dei vari criteri di accettazione e della Definition of Done, accetta o rifiuta la storia.

La modalità con cui la riunione viene organizzata è lasciata a discrezione del team che può organizzarsi nel modo che reputa più efficace. Sebbene il **PO** sia il responsabile

ultimo per valutare il lavoro svolto dal team, se possibile è bene che alla demo partecipino anche altre persone: utenti finali, il cliente che ha commissionato il lavoro, eventuali rappresentanti commerciali o altri stakeholder.

La **Sprint Review** infatti è una **dimostrazione** fatta **con il team** che ha appena completato il proprio lavoro e il cui scopo è quello di raccogliere il maggior numero di informazioni e di feedback sul lavoro svolto. Una tale "apertura" può essere vista con positività dagli "esterni" solo a fronte di una condivisione dei principi e dei valori agili alla base di Scrum. Solo in questo modo, infatti, un errore o una storia bocciata possono essere valutati non come eventi negativi ma come normali componenti di un processo volto a creare il miglior prodotto possibile.

Un esempio di Sprint Review

Sebbene il modo con il quale il team verifica le storie completate sia totalmente demandato al team di sviluppo, riportiamo qui di seguito un esempio di come gestire questa riunione.

La riunione, come spesso accade in Scrum, è coordinata dallo ScrumMaster il quale potrebbe utilizzare una board come quella riportata in figura 16.

Su tale tabella è presente l'elenco delle storie completate che il team porta alla demo; per ogni storia è riportato il titolo, il punteggio stimato durante il planning, l'**owner** e il **tracker** della storia

L'**owner** è una persona del team di sviluppo che si è preso l'incarico di seguirne lo sviluppo, tenendone sotto controllo ogni evoluzione e preoccupandosi che la storia sia completata entro la fine dello sprint. Il **tracker** ha il compito di annotare quello che emerge durante la demo della storia, in modo che sia più semplice apportare modifiche o correzioni alla storia o anche semplicemente per discuterne in un secondo momento.

Nell'ultima colonna della board, si segna con un check **verde** o **rosso** se la storia è stata accettata o meno. Da qualche parte si segna il punteggio effettivamente realizzato, ossia la somma delle storie accettate con successo, in rapporto con il totale previsto durante la pianificazione. Nell'esempio riportato dalla figura 16, il team ha totalizzato 26 punti su 29 previsti, punteggio che rappresenta **Velocity** del team calcolata a posteriori.

Per la presentazione e accettazione delle storie, si segue uno schema come quello riportato qui sotto:

- Lo **ScrumMaster** prende la prima storia dalla lista delle storie finite e porge il cartellino al **Product Owner** affinché egli la legga nuovamente al team; la storia dovrebbe ormai essere nota a tutti, ma è importante ripassarne tutti gli aspetti visto che dal giorno della presentazione al planning è passato un intero sprint. Il **Product Owner** legge sia il fronte del cartellino dove è riportata la descrizione della funzionalità da implementare, che, e soprattutto, il retro con i vari **Acceptance Criteria**.
- A questo punto l'**owner** della storia o un altro membro del team presenta l'implementazione della storia utente e, se è possibile, ne dimostra il funzionamento dal vivo, aspetto quest'ultimo caldamente raccomandato.

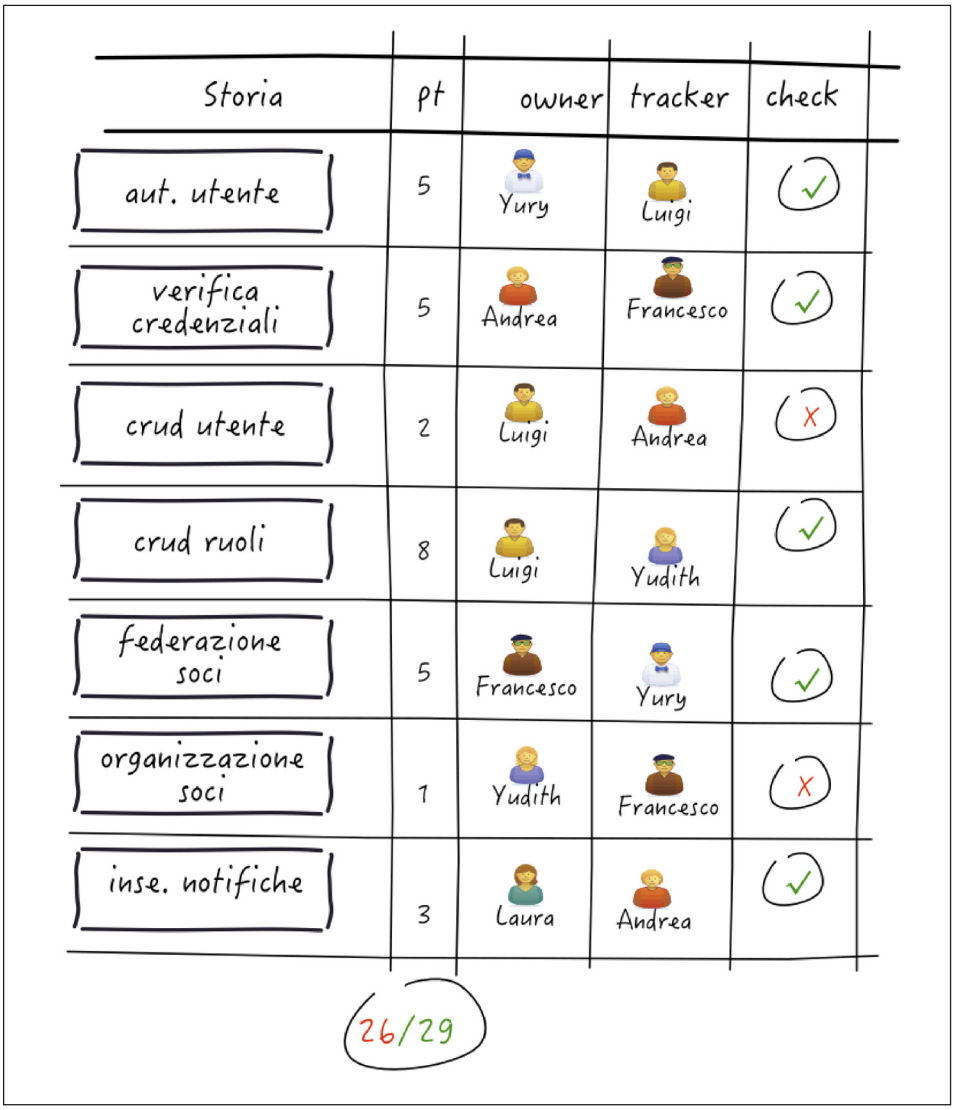


Figura 16. Per la Sprint Review, lo ScrumMaster potrebbe preparare una lavagna con tutte le storie completate in quello sprint e pronte per la demo finale.

- Il **Product Owner** valuta attentamente quanto implementato, verificando che le funzionalità siano state implementate correttamente e che i vincoli siano stati tutti rispettati.
- Qualora il **Product Owner** evidenzi una qualche carenza in un aspetto che era stato indicato nel cartellino della storia, egli procede a **rifiutare la storia**. In caso contrario la **storia viene accettata**. Per tale valutazione sono presi in esame i requisiti

funzionali definiti dagli **Acceptance Criteria**: ogni carenza o presunto errore che il **Product Owner** non abbia inserito in tale lista non potrà avere conseguenze sulla validazione della storia.
- In caso di carenze o aspetti non precedentemente previsti, il **Product Owner** provvederà a preparare una **storia** di **integrazione** o **correzione** e la presenterà in uno dei prossimi **Sprint Planning**. in funzione della sua urgenza o gravità.

Come detto già, il **PO** è il responsabile ultimo per valutare il lavoro svolto dal team, ma è bene che a questa cerimonia partecipino quando possibile anche utenti finali, committenti, commerciali e così via. La **Sprint Review** infatti è una **demo fatta in presa diretta** e se tutti questi stakeholder vengono coinvolti in uno spirito aperto e di comprensione dei valori dell'Agile, eventuali errori o storie non accettate che possano emergere nella **Sprint Review** diventeranno spunti per migliorare il processo e non drammatici eventi negativi che buttano all'aria tutto il progetto...

Cosa fare delle storie non finite

A fine Sprint il team può ritrovarsi con una serie di storie **non Done**: rientrano in questo contesto sia quelle che **non** sono state **terminate** che quelle che, pur terminate, **non** hanno **passato** il vaglio della **Sprint Review**.

È responsabilità del **Product Owner** stabilire **se** la storia debba essere **rimessa in lavorazione**, ossia reinserita nel Product Backlog, e con quale priorità, vale a dire in che posizione di lavorazione.

In alcuni casi il motivo del mancato completamento della storia potrebbe essere da imputarsi a una carenza di informazioni: in tali situazioni il **PO** potrebbe decidere di rimandare la lavorazione in modo da dar tempo al team di raccogliere maggiori dettagli. Il **PO** potrebbe invece ritenere che la storia sia urgente o importante, e quindi decidere di non ritardare oltre la sua implementazione: in tal caso potrebbe inserire la storia nella parte alta del backlog in modo che sia messa in lavorazione nel prossimo sprint. Altre volte infine, proprio sulla base delle informazioni emerse a seguito della lavorazione nello sprint precedente, potrebbe ritenere che tale storia non sia più necessaria, decidendo quindi di rimuoverla definitivamente dal **Product Backlog**.

Nel caso in cui la storia sia reinserita nella lavorazione, fra le altre cose il team deve decidere il punteggio da assegnare all'attività necessaria per completare quello che resta della storia. Il nuovo punteggio potrebbe essere minore, visto che una parte è stata già lavorata, oppure anche maggiore: non averla completata può essere indice di un problema di non facile soluzione.

Alcuni esempi

Non c'è un modo standard di gestire la faccenda e quindi un po' di esempi possono risultare utili per comprendere meglio le varie casistiche. Si consideri per esempio una storia da X punti pianificata nello Sprint 1, storia che per un qualche problema non è stata completata o che non è passata alla demo finale per una qualche carenza.

Nello Sprint 1 si ipotizzi che il team abbia fatto Y punti, e che completi la storia nello Sprint 2. In questo caso si potrebbe ipotizzare che X = Y + Z, dove Z è il numero di punti fatti nello Sprint 2. A questo punto si possono avere differenti possibilità di "contabilizzazione" della storia:

Caso 1

- Sprint 1: Y
- Sprint 2: Z

Questo è il modo più corretto da un punto di vista **matematico** per calcolare la velocità reale del team, ma induce a considerare come già guadagnati i punti Y della storia non finita, il che a volte può essere pericoloso. Inoltre spesso questa matematica non è possibile, visto che nuovamente si tratta di fare stime e valutazioni sulla base di informazioni non certe: quanto è esattamente Y? Chi e come lo stabilisce? Il rischio è quello di spendere molto tempo in attività di scomposizione e analisi: tempo che si potrebbe spendere in modo più redditizio, per esempio, nella implementazione della storia utente.

Caso 2

- Sprint 1: 0
- Sprint 2: X

Questo è il modo più corretto da un punto di vista del processo Scrum: i punti sono **guadagnati** solo quando la storia è finita. Inoltre evita di sprecare tempo in attività di stima e analisi. Questo però porta a un'irregolarità nella velocità, perché al primo sprint si assegnano 0 punti (perdita) e invece si caricano tutti gli X punti nel secondo Sprint: di fatto il secondo sprint guadagna gratis Y punti, indipendentemente da quanto sia Y. Se le storie nello sprint sono piccole, non fa molta differenza; ma, se ci sono storie grandi, il concetto di velocità finisce per perdere di significato.

Caso 3

- Sprint 1: 0
- Sprint 2: Z

Succede spesso, ma è un "errore contabile", in quanto i punti Y vengono ignorati.

In linea generale si consiglia di adottare il secondo approccio, magari suggerendo allo **ScrumMaster** di tenere sotto controllo l'approccio del caso 1 in modo da capire quale sia l'andamento reale della velocità.

Storie accettate con difetto

A volte, durante la demo, capita che in alcune storie presentate, pur complete e funzionanti, contengano piccoli difetti che ne potrebbero impedire l'accettazione finale. Si pensi per esempio al caso di un campo con un allineamento sbagliato, o alla errata

formattazione di una stringa di testo in una maschera di input. Normalmente questi difetti sono risolvibili in pochi minuti da parte del team.

Una strategia che a volte viene messa in atto per gestire queste situazioni è quella di procedere all'**accettazione** della storia **con difetto**: la storia viene accettata, rimandando il lavoro di correzione a una fase successiva alla demo. In genere si tratta di un lasso di tempo collocato tra la fine della retrospettiva e la pianificazione dello sprint successivo: una zona cuscinetto, un buffer, rappresentato da una mezza giornata riservata appositamente per questo tipo di lavorazione.

NOTA: Questa pratica **non** è presente nella **specifica ufficiale** e, sebbene non rappresenti una violazione grave dei principi base della metodologia, è sconsigliabile perché può essere il preludio all'adozione di altre cattive abitudini all'interno del team Scrum: si parte utilizzando solo poche ore per correggere piccoli difetti, ma si finisce spesso per "sconfinare", passando a consumare tempo prezioso che dovrebbe invece essere dedicato all'elaborazione delle storie dello sprint successivo.

Oltre a questo aspetto "temporale", operando in questo modo si introduce un problema legato alla qualità: ogni correzione infatti, per quanto piccola, dovrebbe sempre essere validata in modo formale secondo il processo standard. La correzione dei difetti durante il periodo di buffer porta a chiudere le storie senza che nessuno le abbia verificate dopo le correzioni. In questo caso una soluzione in linea con la metodologia, potrebbe essere quella di eseguire un controllo da farsi a inizio sprint o meglio ancora durante la demo dello sprint successivo.

Con un ulteriore piccolo sforzo si potrebbe rientrare nel processo Scrum: si potrebbe stabilire di rifiutare le storie con difetto, definendo al contempo una attività correttiva, per esempio una storia nuova di integrazione o correzione, che sia poi stimata, pesata e inserita nel Product Backlog per la lavorazione successiva. In questo caso, durante la demo finale ci si preoccuperebbe di verificare la effettiva correttezza delle storie... corrette.

È comunque utile ricordare che uno degli obiettivi primari di un Team Scrum è implementare un **miglioramento continuo**: per questo motivo, più che trovare una strategia di gestione dei difetti, è importante capire come mai alla **Sprint Review** arrivano storie con difetto; per esempio si potrebbero migliorare le attività di pre-verifica prima della **Sprint Review**. Sta al team analizzare i fatti e trovare un modo per migliorare ogni volta.

Valutazione del lavoro del gruppo: Sprint Retrospective

Dopo che il team ha completato le attività di valutazione di quanto prodotto nello sprint, passa a svolgere una valutazione del come ha lavorato, cosa che viene fatta tramite una attività apposita detta **Sprint Retrospective**. Questo momento è estremamente importante, visto che è alla base del processo di miglioramento continuo iterativo e incrementale su cui si poggia la metodologia Scrum e tutta la filosofia Agile.

Data l'importanza di questo argomento, è stato deciso di dedicare alle **Retrospettive agili**, un'intera parte di questo libro, la 5, in cui sono presentate le tecniche da utilizzare e le condizioni che le rendono utili per trovare spunti di miglioramento per l'intero team Scrum. Molte delle tematiche e delle tecniche descritte approfonditamente nella parte 5 sono utilizzabili anche in contesti differenti da Scrum, laddove si decida di provare a dar luogo a un processo di crescita iterativo e incrementale dell'organizzazione, volto a individuare, passo dopo passo, i possibili punti di miglioramento.

Un passo ulteriore

In questo capitolo abbiamo voluto riportare la descrizione delle varie attività che, nel concreto, definiscono lo svolgimento degli sprint come previsti in Scrum. Abbiamo descritto anche qualche situazione che potrebbe discostarsi dalle indicazioni canoniche di Scrum proprio per mettere il lettore a conoscenza di alcuni casi reali che si presentano nella pratica quotidiana e che occorre essere in grado di gestire e di risolvere nel modo più sensato.

Nel capitolo successivo entreremo ancor più nel dettaglio, affrontando un tema cruciale per tutta l'infrastruttura Scrum: le storie utente.

Riferimenti

[CXT] Il formato usato per le storie utente
http://agilecoach.typepad.com/photos/connextra_user_story_2001/connextrastorycard.html

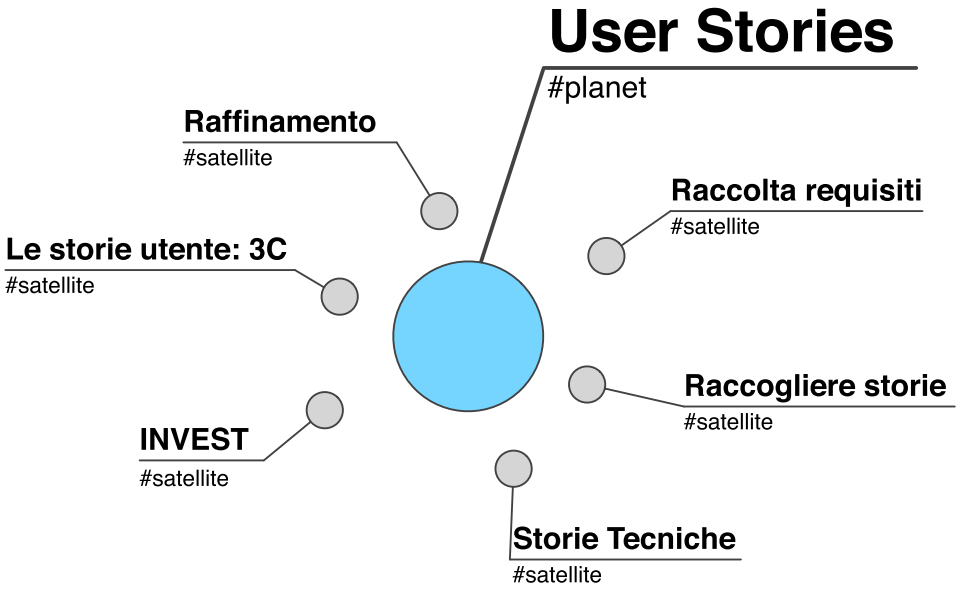
[IS] G. Adzic, D. Evans, *Fifty quick ideas to improve your user stories*. Neuri Consulting LLP, 2014

[PP] La voce "Planning poker" su Wikipedia
http://en.wikipedia.org/wiki/Planning_poker

[DE] La pagina Wikipedia sul Metodo Delphi
http://en.wikipedia.org/wiki/Delphi_method

[AEP] M. Cohn, *Agile Estimating And Planning*. Prentice Hall, 2005

Capitolo 3
Le storie utente



Perché le storie utente?

Abbiamo già avuto modo di accennare che Scrum non fornisce alcuna indicazione sul **formato** da usare per la definizione degli elementi del **Product Backlog**; nella pratica diffusa, però, è ormai universalmente utilizzato il formato delle **User Stories**, preferito sia per la semplicità che per la capacità di adattarsi al processo di lavorazione di Scrum.

Una **storia utente** esprime un **bisogno di business dal punto di vista dell'utente**, tramite un linguaggio naturale, comprensibile sia dal personale tecnico che da utenti e/o esperti di dominio.

Il processo utilizzato per la creazione e le successive modifiche dei requisiti tramite user stories segue un processo incrementale basato su iterazioni e raffinamenti: nella compilazione di una storia si parte dal definire in modo sintetico e ad alto livello le funzionalità che si dovranno implementare per rispondere al bisogno di business in questione. **Esula** dagli obiettivi della storia utente affrontare i dettagli **tecnici**, del **design** o dell'analisi di **dettaglio** della **funzionalità**.

Il ciclo di vita degli elementi del Product Backlog

All'interno del **Product Backlog** il team inserisce tutti gli elementi (**Product Backlog Items**, **PBI**) necessari per il completamento del prodotto; vi si possono trovare quindi **storie** ma anche **bugs**, **requisiti tecnici**, **idee**, **todo items**.

Gli elementi del **Product Backlog** sono classificati in base alla grandezza o al livello di dettaglio: quelli grandi e a grana grossa definiscono un **ampio spettro di funzionalità** e sono spesso detti **Epiche**, proprio per evidenziarne la maggiore dimensione, con le quali si indicano parti principali di funzionalità del sistema (p.e.: "Catalogo prodotti").

Il processo di scomposizione delle epiche porta a qualcosa di più piccolo detto **Features.** Anche le **Features** sono ancora troppo grandi per essere messe in lavorazione all'interno dello sprint: è necessario quindi un ulteriore processo di raffinamento che dà luogo alle cosiddette **storie utente** o **User Stories**. Proprio per evidenziare il fatto che sono pronte per la messa in lavorazione, queste sono dette a volte **Sprintable Stories**, che in italiano si potrebbe tradurre con **storie lavorabili** in uno sprint, anche se questa definizione non è parte della specifica ufficiale di Scrum. La definizione di Sprintable Stories si ricollega al concetto di **Definition of Ready** (**DoR**), ossia una specie di accordo che si stipula a livello di **Scrum Team** per definire cosa può essere messo in lavorazione nello sprint e cosa no.

Le definizioni di **Epic** e **Feature** non sono universalmente adottate, tanto che in alcuni casi si trovano definizioni invertite o basate su termini equivalenti; in questo libro seguiremo questa convenzione, sottintendendo che si tratta appunto di una convenzione.

Come distinguere Epiche, Feature e User Stories?

Un modo per classificare e distinguere **Epiche**, **Feature** o semplicemente **Sprintable Stories**, è quello di dividerle in base al **tempo necessario** per il completamento:

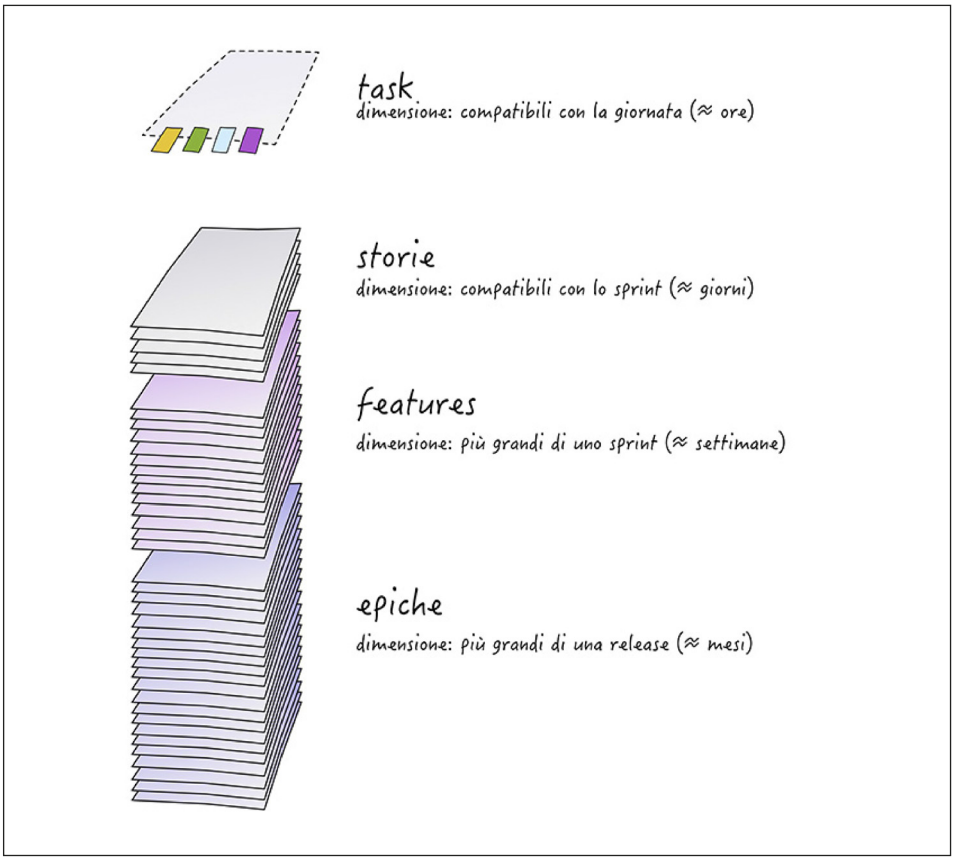


Figura 17. Il backlog è composto da elementi a grana variabile. Benché siano tutte storie, la grana può variare parecchio.

convenzionalmente quindi si dice che una **Epica** richiede alcuni **mesi** di lavorazione, una **Feature** alcune **settimane**, una **User Story** qualche **giorno** di lavoro (figura 17).

Ovviamente queste dimensioni sono da contestualizzare con la lunghezza scelta per gli Sprint: se per esempio le iterazioni fossero di una settimana, il tempo di lavorazione di una **User Story** dovrebbe scendere in modo significativo (poche ore?). Quindi si potrebbe generalizzare dicendo che una **Sprintable Story** non dovrebbe superare una dimensione proporzionale a quella dello sprint, per esempio un quarto o un terzo della durata dell'iterazione; c'è chi si spinge addirittura nel suggerire di restare entro il 10% della durata dello sprint.

Nella parte alta della figura 17 è stata aggiunta una storia ulteriormente scomposta nei suoi **sotto-task**, attività che sono dell'ordine delle poche ore di lavoro. Come si è avuto già modo di accennare, Scrum non fornisce alcuna indicazione sulla necessità di questa ulteriore scomposizione: è una attività comunque utile, ma dato che i task

non danno valore al **PO**, la loro scomposizione normalmente è lasciata in carico al **Dev Team**; i task possono essere visti come una tattica che il team utilizza per raggiungere l'obbiettivo vero, che è finire storie.

Il Product Backlog: in continua evoluzione

Per quanto concerne il **ciclo di vita** dei suoi elementi, il **Product Backlog** può essere considerato come una lista di funzionalità in **continua evoluzione**: elementi grossi e generici (**Epiche** o **Features**) sono smontati in elementi più piccoli, dettagliati con maggiori informazioni e quindi spostati verso l'alto. Contemporaneamente, l'acquisizione di nuove conoscenze sul prodotto da sviluppare può portare a una rivalutazione delle priorità degli elementi posizionati nella parte alta del backlog, elementi che possono essere scambiati con altri più in basso o addirittura rimossi dal **Product Backlog** (figura 18).

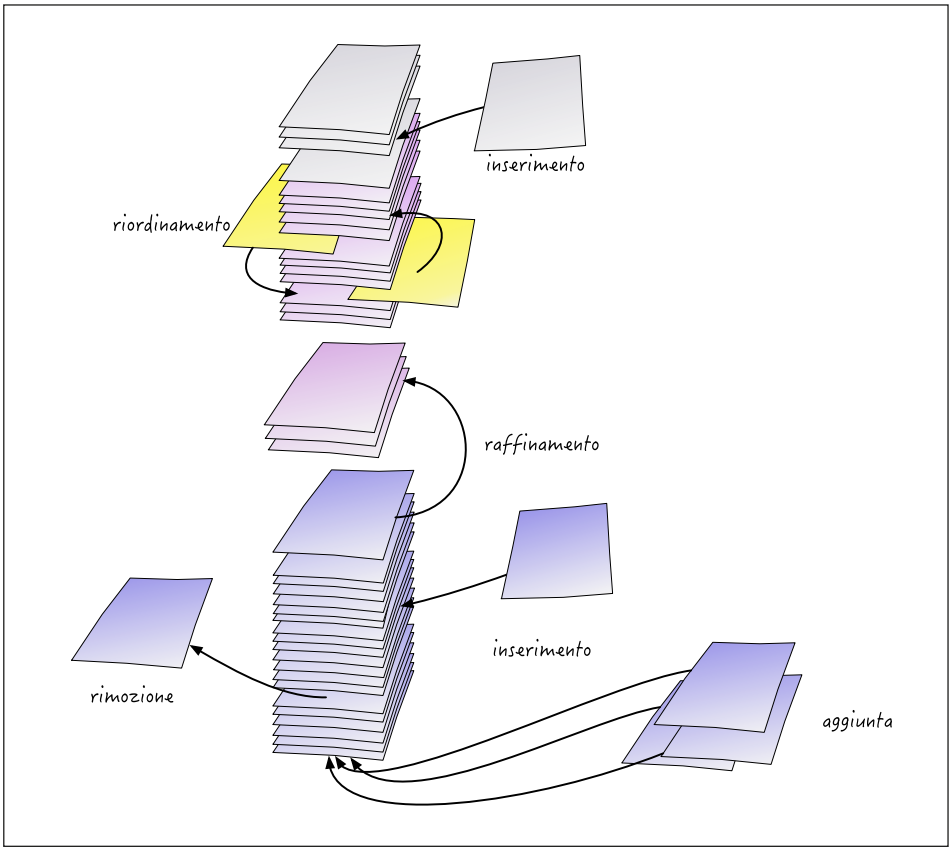


Figura 18. Le storie che compongono il Product Backlog sono in costante evoluzione, per mezzo del Refinement, sia nella loro struttura che nella dimensione e nel posizionamento.

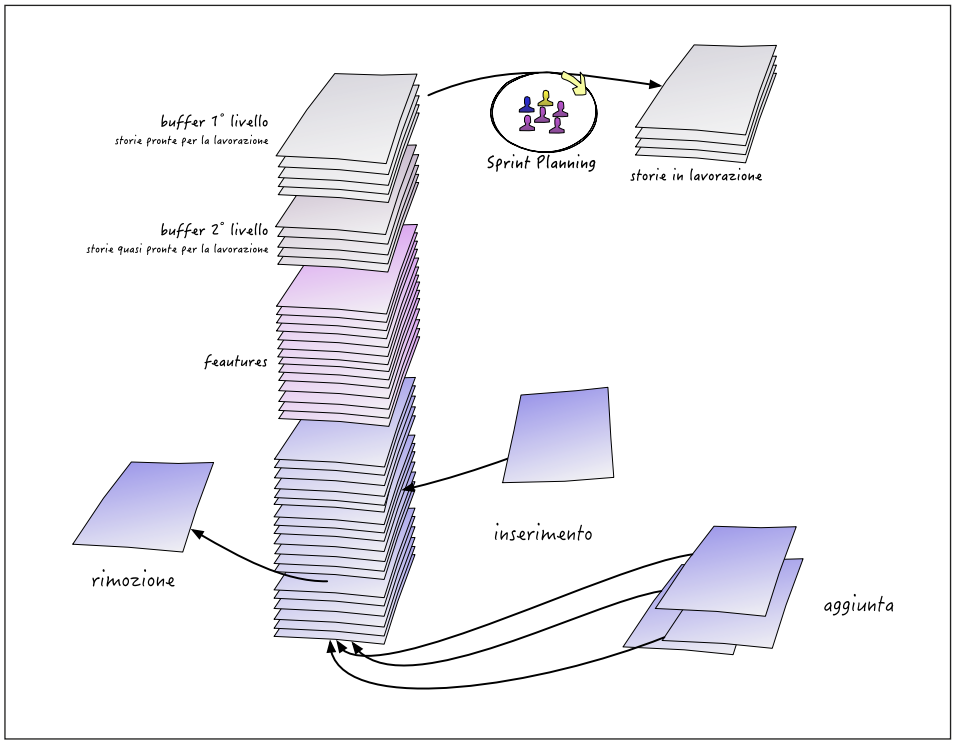


Figura 19. Il processo di alimentazione del backlog è continuativo. La presenza di uno o due buffer permette di ammortizzare eventuali variazioni in accelerazione sulla velocità di lavorazione del team di sviluppo.

Qualora si evidenzi la necessità di aggiungere nuove funzionalità al prodotto finale, può capitare che nuovi elementi siano aggiunti al **Product Backlog**. Dato che non è possibile prevedere la grana e la dimensione di questi nuovi elementi, è necessario mantenere alta la concentrazione su ciò che viene aggiunto al backlog: più un elemento viene inserito in alto, maggior sarà l'urgenza di raffinarlo e prepararlo per la sua messa in lavorazione in uno dei prossimi sprint.

Il buffer delle storie pronte per la lavorazione

Ad ogni sprint, il team di sviluppo "consuma" una parte del **Product Backlog** prelevando le storie lavorabili dalla cima della pila; per questo, il Team Scrum spesso si interroga su quanto tempo sia opportuno dedicare alle attività di raffinamento, ossia quante storie **pronte** per la lavorazione debbano esserci nel **Product Backlog**. Non esiste una risposta definitiva a questa domanda: il team dovrebbe operare in modo da avere sempre un numero di storie lavorabili in grado di garantire il regolare svolgimento del processo Scrum, una sorta di **buffer** contenente le storie pronte per la lavorazione (figura 19).

La dimensione di tale buffer dovrebbe essere compatibile con il **numero medio** di **storie completate** negli ultimi **due** o **tre sprint**. Non è detto che tale numero sia costante, anche perché la **Velocity** è misurata in punti e non in numero di storie. Possono esserci quindi rallentamenti inattesi oppure accelerazioni improvvise, dovute al processo di crescita e miglioramento continuo del team, allo scambio continuo di informazioni e di esperienze maturate nonché alla presenza di un coach.

Per questo il team deve impedire lo svuotamento del buffer, evento che in gergo a volte è detto **pipeline dry** ("prosciugamento del condotto" che alimenta il flusso), senza però cedere alla tentazione di inserire nel buffer un numero troppo elevato di storie. In ogni momento, l'insorgere di nuove condizioni, determinate ad esempio da nuove informazioni, potrebbe evidenziare la necessità di una radicale rivisitazione o peggio dell'eliminazione di storie dal backlog: in questi casi, il lavoro di raffinamento svolto è da considerarsi uno spreco. La dimensione del buffer quindi è direttamente proporzionale al rischio di avere degli sprechi: il buffer di fatto è una forma di "magazzino", che in Lean Production è considerato *waste*.

Il formato delle User Stories

Lo scopo di una **User Story** è rispondere a un bisogno dell'utente in modo chiaro ma senza entrare nei dettagli implementativi: con poche parole, tramite un linguaggio naturale comprensibile sia dal personale tecnico che dagli utenti finali, il requisito è descritto **dal punto di vista dell'utente**, aspetto questo particolarmente importante. Diversamente da altri formati, infatti, una storia esprime una preciso **valore di business** raccontato dal punto di vista dell'utente e non del sistema o del software.

Le direttive ufficiali di Scrum non entrano nel merito del formato e del contenuto delle storie utente, così come non forniscono alcuna indicazione sul loro uso per popolare il **Product Backlog**. La scelta del formato per le storie utente è quindi demandata alle decisioni del team. Un formato efficace delle storie è quello fornito da Ron Jeffries basata sulle tre **C**: **Card**, **Conversation**, **Confirmation** [CCC]. Vediamolo nel dettaglio.

Prima C: Card

Secondo questo formato, le storie sono riportate su un cartoncino (**card**, appunto) di dimensioni medio-piccole, 10 × 15 cm o 13 × 18 cm, organizzato secondo uno schema piuttosto semplice. Sul **fronte** è riportato il **titolo** e la **descrizione** della **storia**. Il titolo deve comunicare in modo chiaro e sintetico il **bisogno** dell'utente. Nella descrizione è riportato il **ruolo** dell'utente a cui la storia si rivolge, la sua **necessità** legata all'azione che egli vuole compiere e l'**obiettivo** nello svolgere una certa azione.

A volte può essere utile inserire un **codice identificativo**, ad esempio l'ID utilizzato nel sistema di tracking, e il **peso in punti**. Sul **retro** del cartellino sono invece riportati i cosiddetti **Acceptance Criteria** (in italiano **criteri di accettazione**), che saranno utilizzati per verificare il completamento o meno della storia.

Per la compilazione del corpo della storia si segue in genere questo schema:

In qualità di <RUOLO>
vorrei che <OBIETTIVO>
affinché <BENEFICIO>

Un esempio di una storia potrebbe essere:

In qualità di utente del sistema di home-banking,
vorrei poter effettuare il login in modo univoco,
affinché tutte le operazioni eseguite siano collegate al mio account.

Come si può notare, **non** sono inserite nella storia informazioni legate al **come** realizzare il flusso operativo, né tantomeno dettagli tecnici pertinenti l'implementazione (figura 20).

Criteri di accettazione

Fra i vari formati utilizzabili per specificare gli **Acceptance Criteria** presenti sul retro del cartellino, spesso si usa un semplice elenco di requisiti che dovranno essere rispettati dalla storia affinché possa essere considerata correttamente implementata. Ecco un esempio relativo a una storia di autenticazione utente:

- se l'utente inserisce username o password errati, verrà visualizzato un messaggio di errore, senza indicare se l'errore è in username o nella password;
- se l'utente inserisce per tre volte consecutive una coppia errata, verrà bloccato il suo account;

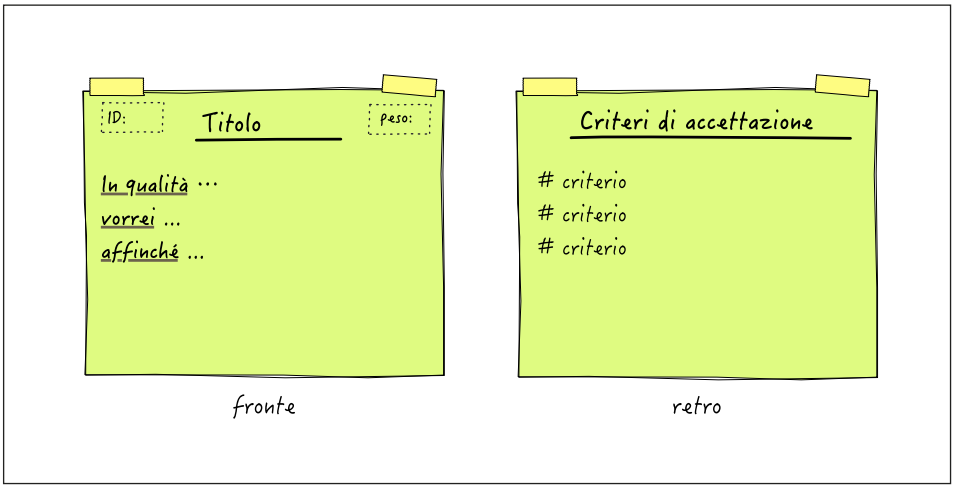


Figura 20. Un esempio di impaginazione dei due lati di un cartellino secondo uno dei formati più popolari.

Figura 21. Un esempio di una storia compilata, con tanto di criteri di accettazione.

- l'utente deve specificare obbligatoriamente se vuole accedere al sistema per effettuare una operazione di consultazione o una dispositiva: nel secondo caso dovrà eseguire il login avanzato (data di nascita, password di secondo livello);
- ...

Seconda C: Conversation

La seconda C del modello proposto da Jeffries è legata al processo con cui i requisiti sono individuati e raccolti. Il cartellino della storia viene utilizzato per la **presentazione** da parte del **Product Owner** al **Dev Team**: in questo momento gli sviluppatori fanno tutte le domande del caso in modo da comprendere i dettagli funzionali. Successivamente, quando la storia verrà presa in lavorazione, verrà arricchita con ulteriori informazioni in modo da chiarire tutte le informazioni necessarie alla sua realizzazione.

Se non ci sono indicazioni prestabilite in merito, il team è libero di scegliere il formato di analisi e documentazione che ritiene più adatto: appunti in un qualche formato, note, diagrammi UML, Use Case Form; a volte invece la decisione sul formato e sulla quantità di allegati da utilizzare per la documentazione è parte delle specifiche di progetto e, in questo caso, conviene inserirle all'interno della **Definition of Done**.

La storia nel formato della **card** deve rimanere **piccola** e **semplice**, perché tutte le informazioni di corredo saranno inserite in un secondo momento. Una celebre battuta dice che, "se non si è trovato abbastanza spazio nel cartellino per descrivere tutti gli aspetti della funzione che si dovrà andare a implementare, il suggerimento è di... prendere un cartellino più piccolo".

Durante la compilazione della storia ci si deve limitare quindi a indicare gli elementi essenziali, come il bisogno utente da risolvere e quali sono le condizioni per stabilire che tale bisogno è stato risolto. Nella storia non si devono indicare le soluzioni implementative; una storia rappresenta invece una sorta di promemoria per **intavolare una**

discussione, in cui far fluire idee, scambiarsi pareri e opinioni; una storia utente è quindi un **impegno a rivedersi**.

Per questo si parla si **conversazione**, che può, anzi deve, aver luogo in tutti i momenti in cui lo si ritenga necessario:

- durante il raffinamento, quando si inizia a prendere visione dello scopo della stessa;
- durante il planning, quando il **Product Owner** risponde a tutte le domande del **Dev Team** circa gli aspetti funzionali della storia;
- durante la fase di implementazione nella **Sprint Execution**: è in questo momento infatti che si potranno approfondire e completare gli appunti presi durante la sua presentazione allo **Sprint Planning**; è durante l'implementazione che si consultano le persone coinvolte o in possesso di informazioni al fine di smarcare gli aspetti legati all'analisi funzionale, tecnica, e di implementazioni varie.
- In pieno accordo con i principi agili, questi incontri dovrebbero privilegiare la comunicazione verbale e di persona: uno dei principi dell'Agile Manifesto è proprio il **face to face**, anche se ovviamente si potranno usare tutti i canali di comunicazione disponibili come le riunioni in videoconferenza, il telefono o la mail.

Terza C: Confirmation

Per essere certi che, al termine del lavoro di implementazione, il risultato finale sia rispondente alle necessità dell'utente finale, durante la creazione della storia si deve operare in modo da garantire la verifica finale. La terza C, la **conferma**, fa riferimento esattamente a questo passaggio. Le storie, come avremo modo di vedere tra poco, devono essere verificabili e questo requisito passa anche dalla definizione di criteri di accettazione chiari e usabili.

Durante la demo, una semplice e rapida verifica della storia tramite la lettura dei requisiti richiesti negli **Acceptance Criteria** unitamente a quanto richiesto dalla **Definition of Done**, rappresenta il modo più rapido per confermare o meno se quanto fatto corrisponde a quanto realmente richiesto.

Le storie utente: come fare un buon INVESTimento

Scrivere una storia utente è un compito piuttosto semplice; scrivere una buona storia utente... un po' meno, anche perché non sempre è chiaro cosa sia una buona storia utente.

Un buon modo per chiarire questo aspetto è far riferimento all'acronimo **INVEST**, che sta per **Independent**, **Negotiable**, **Valuable**, **Estimatable**, **Small** (o meglio della dimensione opportuna) e **Testable**. Di seguito sono spiegati uno per uno questi importanti concetti.

Independent

Composizione e ordinamento del **Product Backlog** sono in continua evoluzione: le storie si devono poter spostare, invertire, inserire o eliminare dalla lista, nella posizione che il **Product Owner** ritiene più indicata per rilasciare sempre il massimo valore.

Affinché questo sia possibile, le storie dovrebbero essere, il più possibile, **indipendenti** fra loro sia da un punto di logico funzionale che tecnico: non dovrebbero esserci dipendenze tecniche.

Non sempre è semplice ottenere questa condizione d'**indipendenza**, dato che, fra due o più storie, vi possono essere propedeuticità funzionali o tecnico-implementative molto complesse; in questi casi si possono applicare strategie che vanno da una riorganizzazione dei contenuti delle storie, per esempio estrapolando la dipendenza in una storia tecnica a sé stante, alla creazione di **oggetti mock** che permettano di completare la storia dipendente. Attenzione che questa soluzione in genere richiede del lavoro aggiuntivo (una storia di integrazione), necessario per rimuovere il mock. La decisione su quale soluzione adottare dovrebbe sempre essere frutto di una discussione di gruppo fra il PO e il team di sviluppo, nel pieno rispetto dei propri ruoli (cosa fare e come farlo).

Negotiable

Le storie dovrebbero essere scritte in modo da catturare l'essenza della funzionalità di business e lasciare spazio per ulteriori discussioni che saranno focalizzate sui dettagli che compongono la storia.

Negoziare vuol dire portare il proprio punto di vista alla storia, proponendo l'aggiunta di qualche dettaglio o la semplificazione di un requisito funzionale, sempre guardando il requisito nell'ambito del proprio ruolo nel team; il **PO** non si deve preoccupare di proporre soluzioni tecniche, così come il team di sviluppo non dovrebbe forzare il comportamento di una funzionalità alle esigenze di una implementazione più comoda.

Negoziazione potrebbe voler dire "alleggerire" una storia, per esempio eliminando o semplificando qualche criterio di accettazione, per permettere a un numero maggiore di storie di essere inserite nel prossimo sprint. Negoziare vuol dire anche che il PO potrebbe accettare la richiesta del team di sviluppo di modificare l'ordine di un paio di storie per risolvere una dipendenza tecnica. Oppure, sempre nello stesso caso di una dipendenza tecnica, il team di sviluppo potrebbe dover simulare il comportamento di oggetti non ancora realizzati, se il PO non ritiene opportuno alterare l'ordine delle storie.

Valuable

Uno degli elementi fondamentali delle storie utente in Scrum è che ogni storia rilasciata dovrebbe aggiungere **valore** al prodotto che si sta componendo: un storia produce valore se, quando viene inserita nel sistema, lo arricchisce di nuove funzionalità che soddisfano uno o più bisogni di business dell'utente.

Non dovrebbero mai essere ammesse storie se non è chiaro lo scopo e l'utilità dal punto di vista dell'utente finale. Ci sono però delle attivita che **non aggiungono valore** per l'utente ma che sono necessarie per il proseguimento del progetto: per esempio l'installazione di un application server, il setup di connessioni per un qualche motore di autenticazione e così via. Tali attività andranno gestite in modo specifico: sono le cosiddette **storie tecniche** delle quali si parlerà poco più avanti.

Estimatable

Le storie dovrebbero sempre poter essere **valutate** in modo da capirne il peso, ossia lo sforzo necessario per la realizzazione.

I **criteri** che possono migliorare il lavoro del team in fase di valutazione sono la **dimensione** e un buon set di **criteri** di **accettazione**. Il primo aspetto aiuta il team a capire meglio cosa debba finire dentro la storia: per questo deve essere piccola, come specificato nel punto successivo, in quanto una **User Story** troppo grande invece potrebbe dar luogo a interpretazioni generiche con un margine di variabilità ampio. Un set di **criteri** di **accettazione** aiuta il team di sviluppo a capire meglio come la storia debba essere implementata.

Un aspetto a cui fare attenzione è quello relativo all'introduzione nel progetto di requisiti la cui stima è difficile per definizione: un esempio tipico è "come utente voglio che il sistema sia più veloce", dove a causa dell'indeterminatezza del testo è impossibile stimare la quantità di lavoro da fare.

In casi come questi conviene riportare la discussione su metriche tangibili, per esempio introducendo un confronto con il sistema attuale, o imponendo un minimo temporale alle risposte del programma.

Small

Anche se le definizioni di "grande" e "piccolo" non sono specificabili in modo oggettivo, una storia dovrebbe essere **sufficientemente piccola** da richiedere una **frazione di sprint** per essere realizzata. Per esempio su sprint di due settimane, le storie non dovrebbero essere più grandi di un paio di giorni. Questo per quanto concerne il tempo di lavorazione. Ragionando sull'effort invece, una buona regola empirica per ridurre al minimo il rischio dice che la dimensione massima in punti di una storia dovrebbe essere intorno a un terzo o un quarto della **Velocity** di Sprint: volendo essere ancora più rigorosi, c'è chi si pone come soglia il 10%.

Detto questo, confezionare **storie piccole** è particolarmente utile perché permette di focalizzare meglio l'attenzione su quello che deve essere realizzato. Storie piccole si spostano più rapidamente sulla **Task Board** e quindi permettono al team di aumentare il numero di cose fatte per ogni sprint, cosa che aumenta le performance complessive come spiegheremo abbondantemente nella parte 4 dedicata a Kanban.

Avere storie piccole aiuta a comprenderne meglio il contenuto e quindi a stimarle (**S** di INVEST), riduce le dipendenze da altre storie (**I** di INVEST) e spesso semplifica le attività di test (**T** di INVEST). Infine storie piccole sono più maneggevoli nel caso sia necessario scomporle, aggregarle o semplicemente modificarne l'elenco dei criteri di accettazione durante il processo di negoziazione (**N** di INVEST).

Spesso, dopo qualche tempo, **Product Owner** e **Dev Team** imparano a produrre storie piccole e più o meno della stessa (piccola) dimensione, tanto da rendere meno necessaria la stima fatta con i punti. Un buon testo che fornice indicazioni utili per imparare a "tagliare" e "smontare" le storie è il già citato libro di Adzic ed Evans [IS].

Testable

La definizione del corpo della storia, i criteri di accettazione e ogni altro dettaglio che riguardano la user story devono permettere di realizzare una storia che sia perfettamente funzionante in modo che poi l'utente o, meglio ancora, un sistema automatico possano **sottoporla a test**. Non dovrà essere lasciate incompleto alcun aspetto della storia pensando poi di tornarci in un secondo momento: l'intera storia, in quanto tale, deve essere **testabile**.

Storie tecniche

Oltre all'implementazione della parte funzionale, nella costruzione di un prodotto è necessario svolgere una serie di attività di preparazione necessarie per il corretto lavoro del team di sviluppo: approntare gli ambienti di lavoro (installare un database, un application server), far funzionare gli strumenti di cooperazione (sistema di versionamento), creare delle utenze di lavoro e così via.

Sono attività queste che **non producono valore** agli occhi dell'utente finale e quindi **non** si possono chiamare **storie utente**; sono però attività necessarie che, proprio come le storie utente, hanno obiettivi molto specifici: per questo sono in genere gestire come storie, anche se in questo caso si chiamano **storie tecniche**.

Le **storie tecniche** sono create e gestite direttamente dal **team di sviluppo** dato che il **Product Owner** non sarebbe in grado di valutarne l'utilità e quindi di definirne la priorità all'interno del backlog: finirebbero per rimanere schiacciate sotto il peso delle altre storie di business. Quando nello sprint è necessario realizzare alcune storie tecniche, al momento dello **Sprint Planning**, il **Dev Team** si impegna per un punteggio di sprint più basso accettando quindi qualche storia utente in meno.

Le storie tecniche sono gestite dal team in modo del tutto analogo alle storie utente di business: sono stimate in punti, spesso sono suddivise in task, sono prese in lavorazione dal team che le gestisce per mezzo di una **Task Board**. Purtroppo a volte il tempo dedicato alla realizzazione di storie tecniche viene percepito dal **Product Owner** come tempo tolto allo sviluppo del prodotto finale, per cui si dovrebbe sempre fare attenzione a come e quante inserirne in ogni sprint.

Le persone del **Dev Team** dovrebbero, nel modo più trasparente e onesto possibile, valutare la necessità effettiva delle storie tecniche; se possibile, si dovrebbe sempre provare a non inserirne in lavorazione; le strategie in questo senso possono essere differenti.

Un modo di affrontare il problema potrebbe essere quello di provare a **trasformare** le storie tecniche in storie utente, **esplicitandone** il potenziale **valore** di **business**, in modo che il **Product Owner** abbia le informazioni e gli strumenti opportuni per comprenderne la necessità. Prendiamo ad esempio la migrazione verso una nuova versione dell'application server: invece di evidenziarne il costo in termini di tempo speso nella realizzazione di una storia tecnica, si potrebbe provare a valutare i costi derivanti dal non fare tale attività, mostrando come il fatto di usare la vecchia versione

potrebbe impedire l'utilizzo di una qualche tecnologia necessaria per implementare una storia utente.

Qualora non sia possibile trasformare una storia tecnica, si potrebbe provare a **inglobarla** sotto forma di **task** all'interno di una storia di business: per esempio la storia tecnica "refactoring dello strato DAO" potrebbe diventare un task della storia utente "modifica i dati dell'utente". La storia ottenuta in questo caso avrebbe un punteggio maggiore dato che contiene un quantitativo maggiore di "cose da realizzare". Nel caso in cui la parte tecnica possa essere utile anche ad altre storie di business, si dovrebbe cercare di "spalmare" questo costo extra anche alle altre storie, in modo da equilibrare il peso delle varie storie.

Indipendentemente dalla soluzione adottata, se possibile, è preferibile sempre distribuire le storie tecniche (semplici, trasformate o inglobate) su più sprint in modo da avere il "costo" delle attività non di business equilibrato e permettere al **Product Owner** di avere a ogni iterazione una buona dose di cose con valore di business.

Come raccogliere le storie utente

Il processo di raccolta delle storie è una attività importante che è differente a seconda che si debba effettuare una prima raccolta delle storie all'inizio del progetto o che invece si debba alimentare il **Product Backlog** a progetto iniziato

La prima raccolta delle storie prende in genere più tempo, tanto che spesso si organizza una sessione di lavoro specificamente per questo scopo, in cui si usano varie pratiche per identificare e modellare le varie storie. In tal senso le due tecniche più utilizzate sono probabilmente lo **User Story Workshop** e lo **User Story Mapping**.

Il primo può essere considerato come una specie di brainstorming in cui ogni membro del gruppo partecipa nella ricerca delle varie funzionalità che si ritiene debbano essere aggiunte al prodotto che si deve realizzare. Lo **User Story Workshop** è di fatto una riunione poco strutturata organizzata secondo la tecnica che si è vista in precedenza: ogni partecipante propone una serie di funzionalità, una per ogni biglietto, e poi eventualmente si procede al raggruppamento delle storie simili; eventualmente ogni gruppo può dar vita a una epica o, nel caso sia più piccola, a una feature. Questo approccio viene detto **bottom-up**, dato che parte direttamente dalle storie per eventualmente procedere all'aggregazione in epiche e features.

Nello **User Story Mapping**, si applica invece un approccio **top-down**: partendo dall'individuazione dei ruoli degli utenti del sistema (la parte "in qualità di" della card), si procede a individuare l'elenco dei bisogni di ogni ruolo utente; questo primo elenco di bisogni andrà a comporre una prima raccolta di **epiche**, raccolta che formerà la prima versione del backlog. Questa tecnica è stata presentata per la prima volta in forma ufficiale da Jeff Patton nel suo libro *User story mapping* [USM].

Lo scopo di questa parte del libro è vedere come in Scrum si gestisce un Product Backlog a regime e quindi questi temi sono stati trattati nella parte 2 del libro sulle tematiche del *liftoff* "dalla visione al prodotto" per un "decollo verticale"..

L'importanza delle storie

In questo capitolo abbiamo analizzato il tema delle storie utente, mettendone in luce l'importanza, il valore strategico e l'utilità nel processo di definizione del Product Backlog e nella esecuzione degli sprint.

Ma la creazione, la valutazione, l'implementazione e la validazione delle storie utente sono responsabilità di persone che in Scrum svolgono le diverse funzioni previste nel processo. Per capire meglio chi deve fare cosa, nel prossimo capitolo si parla, appunto, dei ruoli.

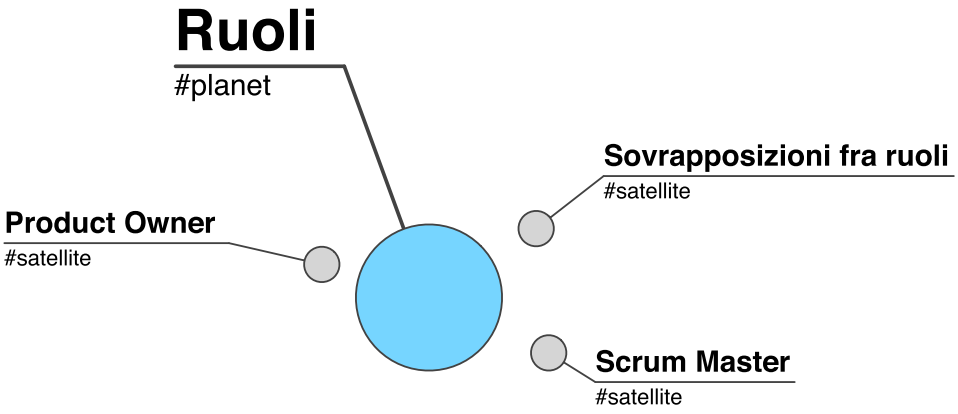
Riferimenti

[CCC] Il formato delle storie basato sulle "tre C": "Card, Conversation, Confirmation"
http://xprogramming.com/articles/expcardconversationconfirmation/

[IS] G. Adzic, D. Evans, *Fifty quick ideas to improve your user stories*. Neuri Consulting LLP, 2014

[USM] J. Patton, *User story mapping*. O'Reilly Media, 2011

Capitolo 4
I ruoli in Scrum



Pochi ma buoni

A più riprese abbiamo accennato alla composizione di uno **Scrum Team**; approfondiamo ora le varie **attività** e **responsabilità** all'interno del gruppo**,** descrivendo i ruoli di **Product Owner**, **ScrumMaster** e **Development Team** (team di sviluppo o **Dev Team**).

Il team di sviluppo e l'organizzazione delle competenze

Il compito del **team di sviluppo** è quello di **implementare** le storie che sono state inserite — o meglio accettate — nello **Sprint Backlog** durante la cerimonia dello **Sprint Planning**. Nella terminologia originale il nome è **Development Team**, anche se alcuni autori, fra cui Craig Larman, lo chiamano **Product Development Team**, per non far sorgere l'errata convinzione che sia composto solo da programmatori.

Per quanto concerne la composizione e le capacità del team di sviluppo, in Scrum si cerca di promuovere la crescita delle persone in modo da distribuire le conoscenze e gli skill; l'obiettivo è quindi quello di creare un team **crossfunzionale**, in cui le persone siano in grado di svolgere tutte le attività della "fabbrica" del software: dalla raccolta dei **requisiti** all'**analisi funzionale**, dalla formalizzazione del **design** all'**implementazione** vera e propria, dalla scrittura della **documentazione** alle fasi di **test**.

I benefici di un team di questo tipo sono molteplici: si riduce il rischio di dover dipendere da una sola persona, si evitano i tempi morti in attesa che questo o quel collega siano disponibili per svolgere un compito specifico, si riduce anche la necessità di dover coinvolgere attori esterni, competenti di tematiche particolari. Questa configurazione permette inoltre di distribuire le conoscenze di dominio, tecnologiche o funzionali, riducendo la necessità di consumare molto tempo in attività di allineamento, documentazione, condivisione delle conoscenze.

Da tenere presente che, nonostante un team crossfunzionale sia più efficiente di un gruppo di persone che lavorano in dipartimenti diversi, se non c'è una ridondanza di competenze si otterrà comunque un schema a **waterfall interno** al team, il che è un'antipattern di Scrum. Per questo motivo una definizione più efficace è quella che Craig Larman introduce quando parla di **Feature Team**, dove è forte il concetto di multidisciplinarità: "focus on multiple specialisations" [FT].

In contrapposizione, si trova il classico schema in cui il gruppo è composto da persone con **competenze specifiche** e **verticali**: gli analisti sanno fare l'analisi, i programmatori la implementano con il codice, altri parlano con il cliente e così via. In questo modello, a volte detto **a silos verticali**, le persone, prima di essere parte di un team di progetto, fanno parte di una divisione o sotto-organizzazione: la loro sorte, carriera, successi e insuccessi sono legati ai successi e insuccessi della divisione di appartenenza e non del progetto in sono coinvolti.

I limiti del modello verticale

Questo modello organizzativo, in maniera sempre più evidente, si sta dimostrando **inefficiente** perché inadatto a risolvere i problemi delle organizzazioni moderne. Molto

spesso dà vita pericolosi colli di bottiglia nel processo di lavorazione: per esempio, se gli analisti sono impegnati al massimo, non potranno prendersi in carico altre attività, cosa che invece sarebbe possibile se i programmatori fossero in grado di contribuire alla raccolta dei requisiti.

Non permette lo **scambio** di **informazioni**, dato che le persone lavorano per compartimenti verticali, cosa che di fatto ha ripercussioni negative sulla coesione del gruppo. Crea una forte dipendenza sulle persone: il caso tipico è quando si ammala la persona che fa i test e i rilasci e nessuno può vedere il lavoro che è stato realizzato dai colleghi. Richiede un effort maggiore in attività di controllo, gestione e condivisione delle informazioni dato che ogni persona del team tende a vedere il progetto dal proprio punto di vista: ad esempio, gli analisti pensano all'analisi senza preoccuparsi della fattibilità tecnica di certe richieste.

Nonostante le limitazioni che introduce, il modello verticale basato sulla separazione di mansioni e competenze continua a essere fortemente presente all'interno delle organizzazioni, specie in quelle di dimensioni maggiori; ma questo accade più per ragioni storiche — "si è sempre fatto così" — o per inerzia organizzativa, perché cambiare richiede un costo economico e mentale non indifferente. Va inoltre considerato che a volte questo modello si perpetua per una naturale evoluzione: i programmatori più anziani sono diventati esperti di dominio e sono passati con il tempo a seguire aspetti funzionali, lasciando la parte tecnologica.

Un altro mondo è possibile...

Qualora decida di abbracciare i principi dell'Agilità, un'organizzazione strutturata in questo modo dovrebbe abbandonare i team organizzati per competenze verticali che rappresentano infatti un **antipattern fortemente sconsigliato**. Nonostante questa trasformazione sia talvolta vista come troppo impegnativa, è bene tener presente che il passaggio a team crossfuzionali può rappresentare l'obiettivo finale di un **processo evolutivo** da compiere in **modo graduale** e per passi.

Per esempio, il primo step potrebbe essere quello di rendere i team autonomi nel loro lavoro, riducendo le dipendenze dall'esterno eliminando il legame fra le persone del team e la propria divisione di appartenenza: sistemisti, commerciali, esperti funzionali dovrebbero far parte del **gruppo di lavoro** e non prestare le proprie competenze al progetto per un periodo limitato.

Si potrebbero scrivere pagine e pagine sugli argomenti della gestione del cambiamento e delle possibili soluzioni alla strutturazione delle attività in un'azienda: il tema dell'**evoluzione dell'organizzazione** è molto vasto e, sebbene sia strettamente legato al processo di adozione delle metodologie agili come Scrum, non rientra negli scopi di questo libro.

Piccolo è bello

Per quanto concerne invece la **dimensione del gruppo**, è opinione comune che un team — un qualsiasi team, che si debba fare Scrum o altro — sia efficace se la dimensione

non supera le dieci unità. Più precisamente, dimostrazioni pratiche e studi sulle dinamiche di gruppo [TSIZE] indicano la dimensione ottimale in 7 ± 2 elementi. Sotto le 5 persone probabilmente non si riesce a fare massa critica e a innescare un volume di lavoro efficace. Sopra le 9 unità aumentano le difficoltà di coordinamento e di comunicazione tanto da rendere difficili le dinamiche di scambio di informazioni e di auto-organizzazione.

Ovviamente il numero non è una costante universale, ma un parametro empirico: ci sono dei team che, per le personalità coinvolte, possono lavorare anche se superano tale dimensione, mentre in altre squadre 9 persone sono già troppe.

Organizzazione del lavoro

Per quanto concerne l'organizzazione del lavoro quotidiano all'interno dello sprint, il team di sviluppo si auto-organizza, in totale **libertà** e **autonomia**, per svolgere le proprie attività; il gruppo decide come affrontare l'implementazione delle storie utente, stabilendo se sia necessaria una scomposizione in task e se tali task debbano essere di tipo funzionale (scomposizione verticale della storia adatta quando si hanno team crossfunzionali) oppure se la scomposizione debba rispecchiare le competenze del team verticale (task di analisi, design, implementazione, test...).

Dev Team, **ScrumMaster** e **Product Owner** collaborano bilanciando da un lato il rispetto dei propri compiti e responsabilità, dall'altro superando tali restrizioni per il bene comune del progetto e soprattutto del team. Per esempio non è scritto da nessuna parte che un **PO** non possa aiutare a testare le storie, lo **ScrumMaster** a documentare le storie, il **team di sviluppo** a contribuire per la compilazione delle storie e altro ancora.

Il Product Owner

Il **Product Owner** (**PO**) è la persona che fornisce le indicazioni al team di sviluppo in modo che, iterazione dopo iterazione, sia prodotto qualcosa in grado di soddisfare i bisogni di business dei vari stakeholder. Per questo motivo, egli da un lato dice al team cosa deve essere fatto, dall'altro lavora a stretto contatto con utente e cliente per identificare e formalizzare i loro bisogni.

Il **Product Owner** si preoccupa di **cosa** verrà realizzato, mentre il team di sviluppo decide il **come** questo debba essere fatto: il **PO** è quindi il responsabile del **Product Backlog**, del quale stabilisce il contenuto e soprattutto l'ordine.

Non ha alcun potere sulle scelte **tecnologiche**, **architetturali** e **implementative**, che invece sono responsabilità del team di sviluppo. Fanno eccezione, ovviamente, tutti i dettagli tecnici che sono parte di specifiche richieste del cliente, il cui rispetto rientra quindi fra le responsabilità del **Product Owner**.

Uno dei compiti più difficili che egli deve svolgere è quello di individuare il **valore** di ogni funzionalità richiesta dall'utente e implementata dal team: il valore diverrà il criterio con il quale sarà definito l'ordine delle attività e del rilascio.

All'interno di uno Scrum Team, il **PO** si preoccupa quindi di impostare lo sviluppo del progetto in modo da rispettare gli aspetti contrattuali del cliente: rispetto del budget, delle scadenze, delle date di eventuali rilasci.

Se necessario, aiuta l'organizzazione e il cliente a condividere una **visione comune** e quindi a stipulare un accordo economico che sia maggiormente compatibile con i principi agili; si parla infatti dei cosiddetti **contratti agili**. Si preoccupa quindi di definire e aggiornare insieme al contraente la roadmap dei rilasci affinché sia compatibile con gli accordi commerciali stipulati e consenta quindi una sufficiente sostenibilità economica (flussi di cassa, rispetto dei contratti o altro).

Deve avere una chiara **visione** del **mercato** e per questo motivo interagisce quotidianamente con clienti esistenti ma anche con quelli nuovi o potenziali. Intervista *buyers* del prodotto per capire i loro problemi e bisogni. Ne sintetizza i risultati a garanzia di scelte di business basate e supportate da informazioni certe, vale a dire dal mercato e non da opinioni volatili, e ne cura l'attuabilità. Il ruolo di **Product Owner** può essere quindi considerato quello più vicino agli aspetti di business del progetto.

Non è detto che il **PO** svolga di persona tutte queste attività; alcune può delegarle, come ad esempio le interviste con il cliente o l'utente finale.

Il suo ruolo è anche quello di **supportare** l'azienda nel seguire e gestire attività legate al business; per questo egli si preoccupa di:

- seguire la **progettazione** della soluzione e guidare la sua **validazione** sul mercato;
- guidare e verificare il **soddisfacimento** dei **requisiti** di alto livello e quindi dell'iniziativa di business;
- supportare la strategia migliore per il **go-to-market** di prodotti e iniziative di lancio, aiutando il marketing e i responsabili della comunicazione;
- facilitare **decisioni** (informate) riducendo il dibattito tra opinioni (personali) e portando sul tavolo "fatti di mercato";
- bilanciare le attività tattiche con quelle strategiche facendo sempre riferimento alla **visione** e alla strategia dell'azienda;
- conoscere a livello qualitativo la **concorrenza** e approfondirne lo studio con strumenti o metodologie appropriate se e quando opportuno.

Cura e continuità sulla ownership del backlog

Affinché tutti gli stakeholder (interni ed esterni) abbiano chiaro il lavoro da svolgere, il **PO** gestisce il **Product Backlog** in modo trasparente e ne mantiene visibile a tutti gli interessati il contenuto e l'ordine.

Data l'importanza di questo compito, spesso il **Product Owner** si avvale di collaboratori con ruolo analogo (**vice-PO** o Product Owner subordinati) in modo da garantire continuità e supporto per la cura degli aspetti e degli interessi dal punto di vista del business: si dice che si avvale dei cosiddetti **PO proxies** sul lato business.

Parallelamente, per garantire la miglior qualità possibile del risultato finale, quando sia necessario affrontare particolari temi, si avvale di consulenti con competenze

specifiche (**Subject Matter Expert**) per raccogliere e confezionare le informazioni, anche se poi la codifica finale di tali informazioni in termini di storie è fatta con i membri del team di sviluppo.

Il lavoro di cura del backlog è quindi spesso un lavoro di gruppo, anche se è il **Product Owner** a esserne il **responsabile ultimo**: ogni decisione in merito dovrà essere da lui vagliata e autorizzata.

Le responsabilità del PO

Come sintesi di quanto esposto nei paragrafi precedenti, di seguito sono riportate alcune **responsabilità** tipiche di un **Product Owner**:

- avere la **vision** del prodotto;
- massimizzare il **ritorno sull'investimento** mediante il rilascio continuo di funzionalità di valore più che con la realizzazione di tutte le funzionalità del prodotto;
- essere la voce del **cliente**;
- definire la **roadmap** dello sviluppo e dei rilasci;
- definire gli **obiettivi** di progetto;
- definire le **metriche** di successo;
- gestire il **Product Backlog**;
- partecipare alle **cerimonie** (allo Sprint Planning, alla Sprint Review e alla Retrospettiva);
- definire (o accettare) le **storie**;
- lavorare a stretto contatto con il **team di sviluppo**;
- gestire le relazioni con gli **stakeholder**;
- gestire o avere una chiara idea del **budget**;
- gestire il **market** & **research**;
- avere **vision** sul **breve periodo** ma anche sul **lungo**: se manca la visione di lungo periodo, fa lavorare il team sprint per sprint ("sembra di essere in una catena di montaggio"); se manca la visione sul breve, potrebbe mancare la motivazione a fare le cose,("non c'è fretta, tanto il progetto finisce fra due anni...").

In riferimento alla figura del **Product Owner**, alcune delle responsabilità sono decisamente imprescindibili dal ruolo; altre invece non è detto che ricadano sempre sotto il suo controllo diretto. In aziende di dimensioni medie e grandi o con un'organizzazione funzionale, il **PO** potrebbe **non** avere, ad esempio, responsabilità diretta della **roadmap** o occuparsi in prima persona della gestione di **market** & **research**.

Lo ScrumMaster

Lo **ScrumMaster** (**SM**) può essere considerato il **coach** o il **facilitatore** il cui scopo è quello di far procedere al meglio il lavoro all'interno del gruppo. Egli è l'**owner del processo** Scrum, nel senso che si preoccupa — ma non impone — che il gruppo segua le **indicazioni** e le **regole** del processo, che i ruoli siano rispettati, che le pratiche siano eseguite secondo i principi "canonici" e che i vari artefatti (User Stories, Burn-Down

Chart, Sprint Backlog e Product Backlog) siano sempre compilati e gestiti in modo corretto.

Ma, prima di tutto questo, allo ScrumMaster sta a cuore che il team usi Scrum in modo da essere **efficiente piuttosto che fedele alle regole**. Se, ad esempio, il rispetto rigido dei ruoli si ripercuote sul gruppo al punto che non si lavora bene, lo **SM** proporrà di modificare l'implementazione dei ruoli in modo che siano più produttivi.

Anche se non fa parte della definizione ufficiale del suo ruolo, lo **SM** spesso cura direttamente tutte le informazioni utili di progetto e ne favorisce la creazione e la condivisione da parte di tutte le persone del team. Per favorire questa condivisione delle informazioni di progetto, si avvale di strumenti utili alla "diffusione" di tali conoscenze: si tratta dei cosiddetti **information radiator**s quali il poster con l'**Elevator Pitch** o la **Vision Board**, la **Skill Matrix** del team, il **Working Agreement** e altro ancora.

A parte queste attività "organizzative", il suo ruolo è molto più complesso e delicato, e riguarda la **gestione delle persone** all'interno del gruppo, la crescita continua e altro ancora.

Il lavoro dello ScrumMaster. Parte prima: lavorare al servizio del gruppo

Fra le molte attività assegnate al ruolo dello **SM** si trovano la **rimozione** degli **impedimenti**, intesi come impedimenti di organizzazione e non problemi di ordinaria amministrazione tipo trovare un cavo di rete, la **protezione** da **perturbazioni** esterne, ma anche permettere che il gruppo possa lavorare al meglio, per esempio predisponendo un **ambiente** di **lavoro piacevole** o fornendo gli **strumenti necessari** per lavorare in modo efficace.

Lo ScrumMaster si premura che tutti abbiamo le **conoscenze** per poter svolgere il proprio lavoro e, nel caso, si attiverà per far sì che eventuali lacune siano colmate, per esempio organizzando piccoli corsi di formazione oppure facilitando lo scambio di informazioni fra i membri del team con momenti di confronto, pair programming o altro.

Un compito che spesso viene dato in carico allo ScrumMaster è quello di **proteggere** il team dalle perturbazioni esterne, come per esempio le continue interruzioni per partecipare a riunioni oppure le attività extra-backlog — quelle non pianificate durante lo Sprint Planning — legate o meno al progetto in questione. In tal senso, si dice spesso che uno **ScrumMaster** dovrebbe essere un **uomo di management** — è una delle domande dell'esame di certificazione di Scrum Alliance — o comunque in grado di parlare con le alte gerarchie dell'organizzazione al fine di prevenire gli impedimenti o anche semplicemente per riportare le conseguenze di eventuali "azioni di disturbo".

Per descrivere questa interpretazione del lavoro dello **ScrumMaster**, è spesso utilizzata una metafora molto popolare per descriverne compiti e responsabilità: egli è il **servant leader** del gruppo, ossia una **guida al servizio** del gruppo. Secondo questa definizione lo **ScrumMaster** è un **leader collaborativo**: non è l'eroe che guida il suo gruppo verso la meta, ma è colui che vive dentro il gruppo, lo motiva e lo stimola; rappresenta il punto di riferimento in caso di dubbi o incertezze; non comanda, ma mostra errori

o cose fatte bene; cerca di riportare il lavoro nell'ambito del corretto modo di agire. In questo senso è quindi al servizio del gruppo.

La metafora del *servant leader* deriva dal lavoro fatto da Robert Greenleaf negli anni Settanta, e prende spunto dalla storia raccontata nel libro di H. Hesse *Il pellegrinaggio in Oriente* (1932, prima ed. it. 1973) [AJE], in cui un gruppo di personaggi intraprende un lungo viaggio a piedi verso un idealizzato "Oriente", viaggio che si svolgerà sia nello spazio che nel tempo. Uno di questi viaggiatori porta con sé il suo servitore, il quale, grazie ai suoi modi gentili ma concreti, finisce per influire sullo spirito di gruppo e sulla coesione dei viaggiatori, facendoli agire come una vera e propria squadra.

Il lavoro dello ScrumMaster. Parte seconda: essere il coach del gruppo

Secondo la visione del ruolo dello **ScrumMaster** vista nel paragrafo precedente, lo **ScrumMaster** è un difensore del team, per il quale svolge una serie di azioni per il bene stesso delle persone del team: dalla compilazione degli information radiator, all'erogazione di corsi di formazione, alla protezione del gruppo dalle perturbazioni esterne. Secondo questa interpretazione, il lavoro dello **ScrumMaster** rischia di ridursi a una specie di "quasi ordinaria amministrazione", attività senza dubbio utile, ma che probabilmente non ne giustifica lo stipendio. In realtà, una parte molto importante del lavoro dello **ScrumMaster** è quella finalizzata al **miglioramento** del team tramite il coaching sulle persone e sul gruppo.

Una buona metafora che permette di comprendere al meglio il ruolo dello SM è quella che si riallaccia al "mestiere" di genitore, in cui babbo e mamma si preoccupano di passare ai figli quei concetti fondamentali necessari per vivere sani e felici; nel corso degli anni, guidano, bilanciando sapientemente regole e libertà, alternando momenti in cui faranno da **insegnanti**, altri in cui saranno **mentori,** altri in cui **proteggeranno** il figlio da situazioni che non è ancora pronto a gestire.

Essi sanno che devono proteggere i figli dai pericoli, senza per questo impedire loro di fare quelle esperienze, prima in un ambiente protetto, poi sempre più liberamente, che gli permettono di crescere e rendersi indipendenti. Sanno che i figli potranno commettere degli errori, per cui si preoccupano prima di tutto che questi errori non siano dannosi per la loro salute mentale e fisica. Sanno anche che fare il genitore è un compito difficile, che anche i genitori devono imparare a farlo: sperimentazioni, valutazione delle conseguenze, confronto e correzioni sono tutte azioni che devono far parte del percorso di crescita di un babbo e di una mamma così come di un figlio.

Il coaching come strumento per la crescita e il miglioramento delle persone

Questa visione della vita è molto vicina a quella che uno **SM** dovrebbe avere con il suo team: il suo lavoro è prima di tutto focalizzato a far crescere le persone nel gruppo, proteggendole quando non ancora in grado di farlo da sole, in modo che esse possano lavorare nel modo migliore possibile. In una parola dovrebbe essere il **coach del gruppo**. Certamente dovrebbe fare molte delle cose elencate nel paragrafo

precedente, anche se sempre nell'ottica di consentire al team di rendersi autonomo, capace di affrontare le diverse problematiche che si possono incontrare all'interno di un progetto.

Spesso si trova scritto che uno **SM** dovrebbe favorire quindi l'**empowerment** del team. Questa è una definizione corretta ma alla quale manca una parte. Un bravo **SM** dovrebbe puntare all'**emancipazione** del gruppo, che è il vero modo per farlo crescere: fare *empowerment* di fatto è un modo per trasferire le conoscenze dello **SM** e del suo modo di intendere il lavoro di gruppo; ma non è necessariamente detto che tale modo di lavorare sia per sempre adatto al gruppo o ai nuovi contesti dove il team si muoverà.

Fare empowerment quindi favorisce certamente la **diffusione della conoscenza**, ma **impedisce l'evoluzione**; è spesso un freno che non consente al gruppo di sperimentare nuove soluzioni e nuove idee.

Un genitore che replicasse nel figlio il suo modo di intendere la vita, non lo preparerebbe a fronteggiare i nuovi scenari e a cogliere le differenti opportunità che si possono incontrare in un mondo che cambia continuamente. Un bravo genitore, così come un bravo coach, o un bravo leader, deve favorire l'iniziativa dal basso, deve creare le condizioni per rendersi non più indispensabile.

Per capire se egli sta facendo un buon lavoro, si potrebbe prendere l'elenco delle attività del paragrafo precedente, e capire quanto il team sia in grado di eseguirle in autonomia, quanto sia capace di proporre delle varianti o di introdurre nuove soluzioni.

Nelle discipline psicologiche esiste un approccio teorico-pratico che si chiama **Analisi Transazionale**, all'interno del quale si affronta il tema dell'attitudine genitoriale che molte persone hanno. Secondo questa teoria, l'atteggiamento dello **ScrumMaster** appena descritto potrebbe essere rappresentato da quello che Eric Berne (padre della Analisi Transazionale) descrive come il Genitore Normativo Positivo (GN+). Per chi fosse interessato ad approfondire queste tematiche consigliamo la lettura degli articoli pubblicati su MokaByte qualche tempo fa [AT] dove è possibile trovare molti approfondimenti e riferimenti ai testi originali di Berne.

Accanto alla metafora del Servant Leader, e accanto a quella del GN+, alcuni autori [HOST] ne propongono un'altra che si ricollega al concetto di **host leadership** [HL].

In questo caso l'**Host Leader** del gruppo può essere visto come il padrone di casa organizza e gestisce una festa in casa: egli si preoccupa che tutto funzioni, che cibo e musica siano adeguati, controlla che nessuno esageri disturbando la festa ma che tutti partecipino e si divertano. Probabilmente lui è il punto di riferimento per le persone ma non fa pesare questo suo ruolo, conscio che la festa funziona meglio se tutti si divertono e se tutti sono allo stesso livello. Quindi è il promotore della festa quando si comincia, ma poi lascerà che la serata segua il suo corso, partecipa e si diverte con gli altri. In determinate situazioni egli sarà il garante delle regole della festa, regole che però sono definite dal gruppo; così come saranno individuati dal team i criteri per stabilire se una festa è riuscita o no.

Caratteristiche di uno ScrumMaster

Specialmente quando si deve dar vita a un nuovo team Scrum, ci si domanda spesso quali siano le **caratteristiche caratteriali** e **professionali** di un buon **ScrumMaster**. Certamente deve avere una spiccata propensione all'ascolto e alla gestione dei rapporti interpersonali all'interno del gruppo, deve essere in grado di riconoscere ogni variazione negli equilibri all'interno e all'esterno del team. Deve saper intercettare i bisogni di tutti e farsi portavoce verso l'esterno delle necessità del gruppo.

In alcune occasioni potrebbe essere utile, se non necessario, che lo **SM** faccia "pesare" la sua esperienza, "suggerendo" una soluzione a un determinato problema. Ma nella maggior parte dei casi, si preoccuperà di stimolare la discussione per consentire al gruppo di proporre e valutare le varie opzioni e le conseguenze delle varie scelte.

Quando si fa portavoce verso l'esterno, deve farlo con i modi giusti affinché sia ascoltato: quando si parla di protezione del team dalle conseguenze delle azioni intraprese dal management, lo fa sempre in modo da mettere in evidenza le implicazioni delle decisioni dei capi.

Aspetto pratico molto importante è legato al **come si sceglie** (si nomina? si incarica? si elegge?) lo **ScrumMaster** all'interno del gruppo. Buona norma è lasciare che il team si organizzi scegliendo in autonomia il proprio **SM** e supportandolo nel suo percorso di crescita professionale. Spesso quindi lo **SM** è una persona dello staff tecnico o un programmatore che mostra di avere una propensione per la gestione delle persone.

ScrumMaster: formarlo all'interno o cercarlo all'esterno?

Chi svolge questo lavoro deve avere una preparazione specifica: quando un membro interno allo staff decide di intraprendere questo tipo di percorso, è importante quindi che l'organizzazione lo supporti, sia con attività di formazione dal momento che esistono corsi da coach molto efficaci, sia consentendo la pratica di tale attività, per esempio facendo in modo che possa svolgere questo mestiere con più team. A tal proposito si dice spesso che un bravo **SM** può seguire più di un team contemporaneamente; con il tempo, quando egli diventerà più bravo ed esperto, seguirà un solo team.

Qualora non sia possibile identificare una persona con i requisiti adatti o che desideri svolgere questo lavoro, è buona cosa se l'organizzazione si avvale del supporto di uno **psicologo** con adeguate competenze o, meglio ancora, di un **business coach**.

Quando è una persona del **Dev Team** che svolge anche il ruolo di **ScrumMaster** si può andare incontro a uno scenario tipico: finché tutto procede per il meglio, non si evidenziano conflitti; ma quando il progetto attraversa una fase di difficoltà, la coabitazione dei due ruoli (**SM** e **sviluppatore**) nella stessa persona può essere causa di problemi. Si pensi per esempio al caso in cui il team di sviluppo si trovasse in ritardo nella consegna di una qualche funzionalità: in quel momento, proprio quando il ruolo del **coach** potrebbe aiutare a risolvere i problemi, lo **SM** difficilmente riuscirebbe a mantenere il distacco necessario per fare il suo lavoro di coach e probabilmente si metterebbe a lavorare come sviluppatore al fianco dei colleghi.

In conclusione: pochi ruoli e molta collaborazione

Scrum definisce in modo molto preciso tre differenti ruoli (Dev Team, Product Owner, ScrumMaster), specificando l'elenco delle attività e delle responsabilità delle varie figure. Il fatto che i ruoli siano pochi e ben specificati serve a dare ordine al modo con cui i membri del team lavorano, stimolando la collaborazione e la discussione.

Questo tipo di organizzazione da un lato si pone come obiettivo la realizzazione di un prodotto di qualità che risponda ai bisogni dell'utente finale, dall'altro persegue il miglioramento del gruppo, una capacità maggiore di condividere le informazioni, la crescita professionale delle persone (singolarmente e come gruppo).

Grazie a questa suddivisione dei compiti si dà vita a un approccio al lavoro co-creativo e collaborativo, fattore essenziale per rompere il modello waterfall. Per esempio fra gli obiettivi del PO, come da vision Toyota, c'è quello di far crescere le persone, piuttosto che puntare solo al raggiungimento dell'obiettivo. In tal senso è vero che il PO funge anche da tramite fra team di sviluppo e committente, ma se svolge questo ruolo in maniera esclusiva, diventando un collo di bottiglia e impedendo la comunicazione fra team e stakeholder, non abilita l'empowerment e la crescita: se il PO è solo un proxy di fatto si sta facendo waterfall...

Riferimenti

[FT] Feature Team
http://www.craiglarman.com/content/feature-teams/feature-teams.htm

[TSIZE] E. Tabbert, "Team size in agile development. A limiting factor?"
http://www.academia.edu/4893042/Team_size_in_agile_development_-_A_limiting_factor

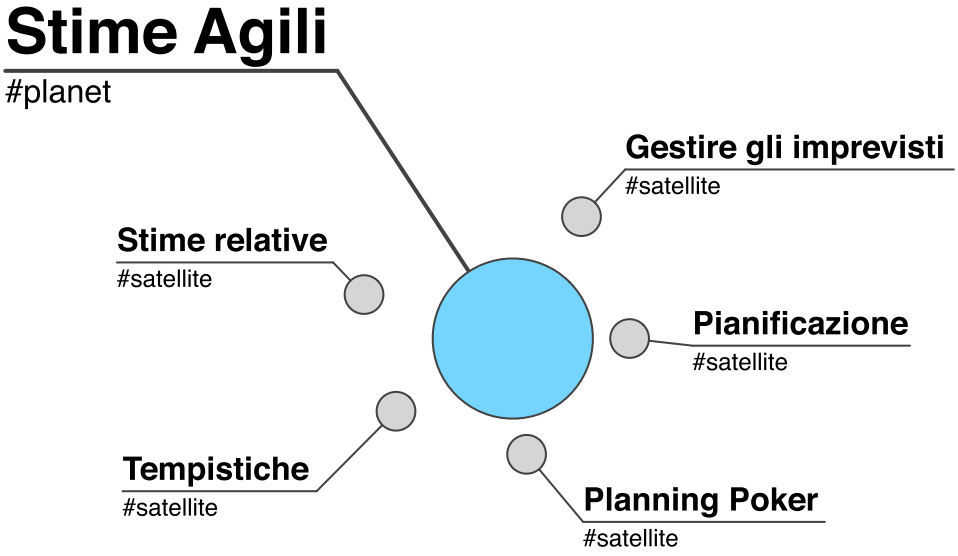
[AJE] La pagina Wikipedia sul libro di Hermann Hesse *Il pellegrinaggio in Oriente* (1932)
https://it.wikipedia.org/wiki/Il_pellegrinaggio_in_Oriente

[AT] G. Puliti, "Aspetti psicologici nella gestione di progetto. I parte: Introduzione all'analisi transazionale nel project management". MokaByte 174, giugno 2012 (e seguenti)
http://goo.gl/aUi7Mr

[HOST] P. Pugliese, "I'm not a servant: I'm a host! A new metaphor for leadership in Agile?".
http://www.infoq.com/articles/host-leadership-agile

[HL] Il sito che illustra il concetto di "host leadership"
http://hostleadership.com

Capitolo 5
Stime agili: cosa sono, come e quando farle



"Ma è vero che in Agile non si fanno le stime?"

Chi si occupa di metodologie Agile avrà trovato sicuramente qualcuno, probabilmente un manager di "scuola MBA classica", che gli ha posto la domanda del titolo. L'argomento è complesso e la risposta non può essere un secco "no" o "sì". In questo capitolo affrontiamo quindi il tema delle **stime in Agile** in maniera diffusa e non superficiale. Per iniziare la nostra discussione, cominciamo con la definizione del termine "stima" fornita dal dizionario Treccani.it:

stima s.f. [der. di *stimare*] – Valutazione approssimata del valore numerico di una grandezza e anche il valore numerico medesimo: *s. per eccesso* o *per difetto*, a seconda che si tratti di una valutazione per eccesso o per difetto.

Fare stime sulla realizzazione di un prodotto significa fare un'**ipotesi** sul tempo, sul costo o genericamente sulle risorse necessarie per completare il lavoro, che nel caso del software si concretizza nello sviluppo di un qualche "manufatto" immateriale ma molto concreto (un'app, un'applicazione desktop, una soluzione informatica enterprise, un sito web e così via...). Si parla di ipotesi perché questo tipo valutazione è funzione di numerosi fattori: alcuni sono conosciuti, calcolabili o ricavabili, e quindi contribuiscono in modo deterministico alla stima, ma altri sono del tutto ignoti o non ricavabili.

Trovare una risposta alla domanda "Quanto ci vuole a fare questo software?" è da sempre un compito difficile ed è per questo motivo che, nel corso degli anni, sono stati messi a punto numerosi sistemi di stima: accanto a strumenti matematici come i **function points** o gli **use case points**, si trovano tecniche basate sull'osservazione empirica come gli **story points** usati in ambito agile, metodi basati sul confronto statistico... e si arriva fino a sistemi più esoterici come il lancio dei **dadi**, la lettura dei **fondi di caffé**, il perscrutare il **volo degli uccelli** al tramonto...

Da non scordare poi l'approccio "diretto", in cui il project manager di turno si affida alla tecnica infallibile della **stima con minaccia**: in questo caso egli chiede con tono minaccioso a un programmatore o a un analista "Quanto tempo ci vuole per fare questo software?" fin quando non ottiene la risposta che desidera, che in genere indica tempi brevi...

Per chi fosse interessato ad approfondire il tema relativo alle stime e al rapporto "pressante" con i propri capi, una lettura interessante è *Death March* di Edward Yourdon [DM] del quale si può avere un estratto sintetico sulla pagina Wikipedia dedicata a questo argomento [DMWiki].

In questo capitolo vedremo come il problema delle stime è affrontato **in un contesto agile**: parleremo di stime agili e di comportamenti in linea con i principi dell'agilità, relativamente alla gestione degli **stati** di **avanzamento**, all'identificazione delle **date** di **consegna** e alla valutazione dell'**effort** necessario per la realizzazione del progetto.

Se è vero che, come più volte sottolineato, in Agile il **cambiamento è benvenuto**, il modo migliore per dimostrarlo è nell'ambito della gestione delle stime, che è il contesto **più sensibile al cambiamento.**

Relativamente al tema delle stime è interessante notare come stiano fiorendo da diverse parti nuove idee che tendono a ridimensionarne l'importanza o comunque a considerarle in maniera differente da quanto fatto tradizionalmente. Senza dover abbracciare le posizioni radicali di Ron Jeffries e del suo **The NoEstimates Movement** [NEM], una interessante lettura potrebbe risultare *Purpose Of Estimation* di Martin Fowler: l'autore, in questo articolo [MF], sottolinea l'importanza del significato di una stima, del perché in alcuni casi sia utile lavorare per ottenere un numero (numero di giorni, budget necessario) e perché invece altre volte sia poco rilevante.

Stimare in modo relativo

Riprendendo in considerazione quanto detto a proposito dei temi della complessità (Parte 1), lo sviluppo di un prodotto software è un tipo di attività che ricade spesso nel **dominio complesso**, dove l'approccio **per scomposizione** non è efficace: la soluzione di un problema complesso non si ottiene cercando la soluzione delle singole sottoparti del problema di partenza, sempre che sia facile identificare tutte le varie unità costituenti le parti.

Le parti componenti sono infatti interdipendenti fra loro e contribuiscono in modo differente al "peso" totale del problema complessivo in funzione di come sono assemblate e di come interagiscono. Oltre a questo, influiscono anche le componenti del cosiddetto "ecosistema" del progetto (il team, i requisiti, il prodotto finale, il cliente e gli utenti), le quali si influenzano e danno contributi differenti al progetto a seconda del momento, del caso e delle interazioni stesse. Sebbene un prodotto software finito sia un sistema "solamente" complicato perché è un "manufatto" composto da componenti anche molto sofisticate ma la cui dinamica di interazione è nota, **fare software** è un processo che ricade spesso nel **dominio complesso**.

Ciò nonostante, se anche per un solo momento volessimo comunque provare ad applicare questa strategia di scomposizione, smontando e stimando le singole parti, ci troveremmo comunque di fronte a un problema di fondo: la nostra mente da sola non è capace infatti di misurare puntualmente qualcosa senza ausili di riferimento.

Se per esempio ci trovassimo di fronte a un palazzo, un unico palazzo bianco costruito nel mezzo di una pianura o nel deserto, sarebbe veramente difficile provare a dedurne l'altezza: escludendo valori evidentemente non plausibili — non può essere 10 metri, ma nemmeno 2 kilometri — non resterebbe che la strada di fare qualche ipotesi, magari contando il numero di piani e provando a immaginarne l'altezza di ognuno; anche in questo caso ci ritroveremmo con lo stesso problema di fondo, come stimare puntualmente. In definitiva non avremmo sfruttato a nostro vantaggio quello che sappiamo fare meglio: **confrontare**.

Per la mente umana è estremamente semplice ragionare per paragoni e quindi sarebbe molto semplice confrontare l'altezza del palazzo con quella dei palazzi situati nei pressi: si potrebbe quindi affermare che "il palazzo bianco è più basso del palazzo rosso di circa la metà ed è una volta e mezzo più alto del palazzo verde". Possiamo quindi dire

che il nostro cervello è capace anche di stimare qualcosa in modo assoluto, ma in modo molto più impreciso rispetto a una stima relativa.

Nel software si può applicare lo stesso approccio: per un programmatore o per un analista è difficile stimare il tempo necessario per realizzare una determinata parte di progetto, mentre è certamente semplice ipotizzare quanto tempo ci vuole per realizzare una componente B partendo dal presupposto di conoscere con buona approssimazione il tempo necessario per la componente A e procedendo per confronto. In un progetto agile si applica esattamente questo tipo di approccio: si **stima in modo comparativo**.

Stime comparative e cono dell'incertezza

Se si riconsidera brevemente il processo di raccolta dei requisiti in Scrum, si può ricordare che, partendo da un elenco grossolano di cose da fare, si arriva alla definizione di un backlog contenente storie di grana variabile: in alto sono posti elementi la cui dimensione (piccola) è compatibile con la messa in lavorazione, mentre, a mano a mano che si scende verso il basso, gli elementi crescono di dimensione e diminuisce il loro livello di dettaglio. Come si è già avuto modo di vedere quando si è parlato di planning poker e di stime comparative (figure 11 e 12), il processo parte dalla individuazione

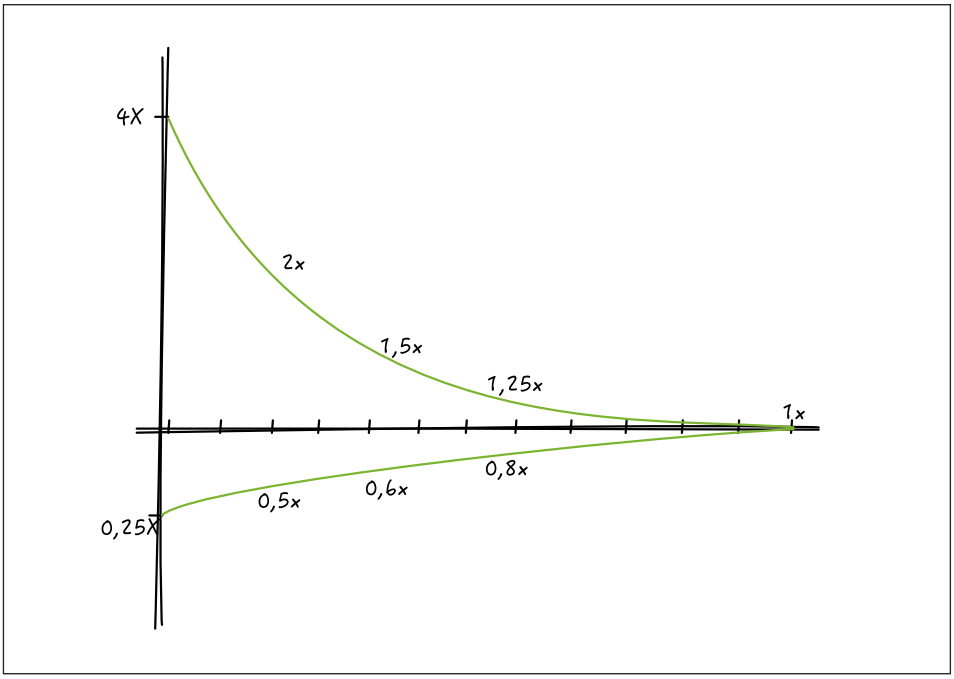


Figura 22. Diagramma detto del "cono dell'incertezza": partendo da uno stato iniziale in cui la variabilità delle stime passa da −1/4 al quadruplo, man mano che il tempo passa e le informazioni arrivano, si può ridurre questa forbice.

della **storia campione** o **storia di riferimento** e dal confronto con le altre storie del Product Bakclog.

Quando si è scelta la storia di riferimento, si sono fatte due assunzioni: quale storia scegliere e che tale storia sia lavorabile in un lasso di tempo piccolo. Sono entrambe **ipotesi** — stime, appunto — che dovranno essere poi confermate dal lavoro sul campo, grazie alla verifica empirica. In questa fase del progetto infatti non è importante, né sarebbe possibile, stabilire con certezza quanto tempo sia necessario per svolgere una storia da 3 punti o una da 21. Conta invece la **stima relativa**: ossia considerando la storia campione da 3 punti, se una storia è più grande potrà essere da 7 o 9 punti. Se molto più grande sarà da 21. Il team con il tempo impara ad adattare le proprie valutazioni mantenendo questi rapporti di proporzionalità.

Col tempo il team imparerà a stimare le storie in modo più preciso; il significato di "3 punti" potrebbe cambiare iterazione dopo iterazione e per questo è bene **non convertire** i punti in ore: con il tempo, iterazione dopo iterazione l'efficienza del team potrebbe aumentare o diminuire; l'equivalenza "3 punti = 1 giorno" potrebbe non essere più vera.

Un interessante aspetto che emerge nei team che adottano questa tecnica è l'insorgere di un clima più rilassato e trasparente durante il processo di stima. Quasi subito viene meno l'impulso a "barare" sui numeri in un senso o nell'altro, sia perché il sistema si auto-adatta a eventuali valori truccati — difatti non si usa il tempo, ma i punti — sia perché il team stesso percepisce gli effetti negativi di un comportamento non trasparente.

Realizzare un prodotto con vincoli di progetto

La gestione di un progetto finalizzato alla realizzazione di un prodotto può essere subordinata a **vincoli** che ne possono influenzare lo sviluppo: tempi di consegna (**fixed time** o **fixed date**), costo complessivo (**fixed budget**), contenuto del prodotto finale (**fixed scope**). Le strategie di intervento sul progetto e su tali parametri fondamentali (aumentare il budget, variare i contenuti del prodotto finale o ritardare le date di consegna), dipendono dal tipo di progetto e di contratto, dai vincoli imposti ossia dai margini di libertà che si hanno a disposizione.

Progetto tutto fisso: fixed date, fixed scope e fixed budget

In questa configurazione, sebbene siano fissati a priori tutti i principali parametri operativi di progetto, è comunque possibile consegnare il prodotto finito rispettando i vincoli fissati. È il caso di progetti software semplici in cui il contesto è noto: per esempio, si deve realizzare un prodotto già fatto altre volte.

Nella maggior parte dei casi invece, sviluppare un prodotto è una attività ogni volta differente per cui risulta estremamente difficile prevedere le implicazioni delle relazioni che legano le varie parti: dipendenze fra le componenti tecnologiche, dinamiche di gruppo, definizione delle funzionalità e relative propedeuticità, variabilità dei requisiti utente e altro ancora.

Per comprendere meglio questo aspetto, può essere utile riconsiderare quanto visto nella parte dedicata alla complessità, in particolare utilizzando il modello interpretativo del framework **Cynefin**, è possibile affermare che in un **contesto semplice** causa ed effetto sono direttamente collegabili nel tempo e nello spazio: in queste situazioni si può prevedere quanto tempo (**fixed time**) e quanti soldi (**fixed budget**) sono necessari per completare un determinato lavoro (**fixed scope**).

Quando invece si lavora in un contesto complesso, purtroppo o per fortuna i fattori che influenzano la riuscita del progetto sono tali e tanti da rendere estremamente difficile rispettare i vincoli imposti (**tempo**, **soldi**, **contenuto**); a volte si finisce per rinunciare alla qualità complessiva del prodotto; oppure si rilassa qualcuno dei vincoli succitati, pena il fallimento del progetto e quindi, in definitiva, si rientra in uno dei casi successivi.

Contesto fixed scope e fixed date

In questo caso si lascia **libero** il **budget** che quindi può essere usato per rispettare la data di consegna e il contenuto del progetto. Questo scenario è spesso di difficile attuazione, sia perché si basa sul fissare due delle grandezze più difficili da prevedere, sia perché nelle organizzazioni raramente è possibile variare il budget a disposizione del progetto dopo che questo è iniziato.

Lasciare libero il budget di fatto significa spendere soldi per **comprare tempo**; questo può essere fatto in vari modi, ma in definitiva vuol dire allocare più giorni-uomo al progetto: o si procede all'inserimento di altre persone nel team di progetto o si opta per lavorare più ore di straordinario. Questa seconda soluzione, contraria alle raccomandazioni dell'agilità (lavorare secondo un ritmo sostenibile continuativo), tipicamente porta a un abbassamento delle prestazioni complessive per via del maggior stress, il cui perdurare può portare al **burn out** del team, forma patologica in cui le prestazioni delle persone si abbassano in modo vistoso.

Per quanto riguarda invece l'inserimento di nuove persone all'interno del team di progetto, si ricordi che anche in questo caso il miglioramento prestazionale, se si manifesta, arriva spesso dopo un iniziale peggioramento a causa delle dinamiche delle persone che lavorano in gruppo.

Questo fenomeno è spiegato da quello che è noto con il nome di **Ciclo di Tuckman**, il quale descrive la crescita di un gruppo di lavoro in quattro fasi: **forming**, **storming**, **norming**, **performing**. Questo modello ci dice che quando si crea (**forming**) un nuovo gruppo, un qualsiasi gruppo, dalla squadra di calcetto al team di sviluppo, prima di arrivare alla fase prestazionale (**performing**) il gruppo dovrà superare gli inevitabili momenti di contrasto (**storming**) interno e la successiva normalizzazione con regole e auto-organizzazione (**norming**).

Interessante notare come il processo di evoluzione verso la fase di **performing** potrebbe richiedere tempi molto lunghi, o addirittura non arrivare mai (configurazione statica in **forming** o in **storming**), a seconda di come è stata gestita la formazione del gruppo. Ogni nuova persona che si aggiunge al team deve imparare cosa e come

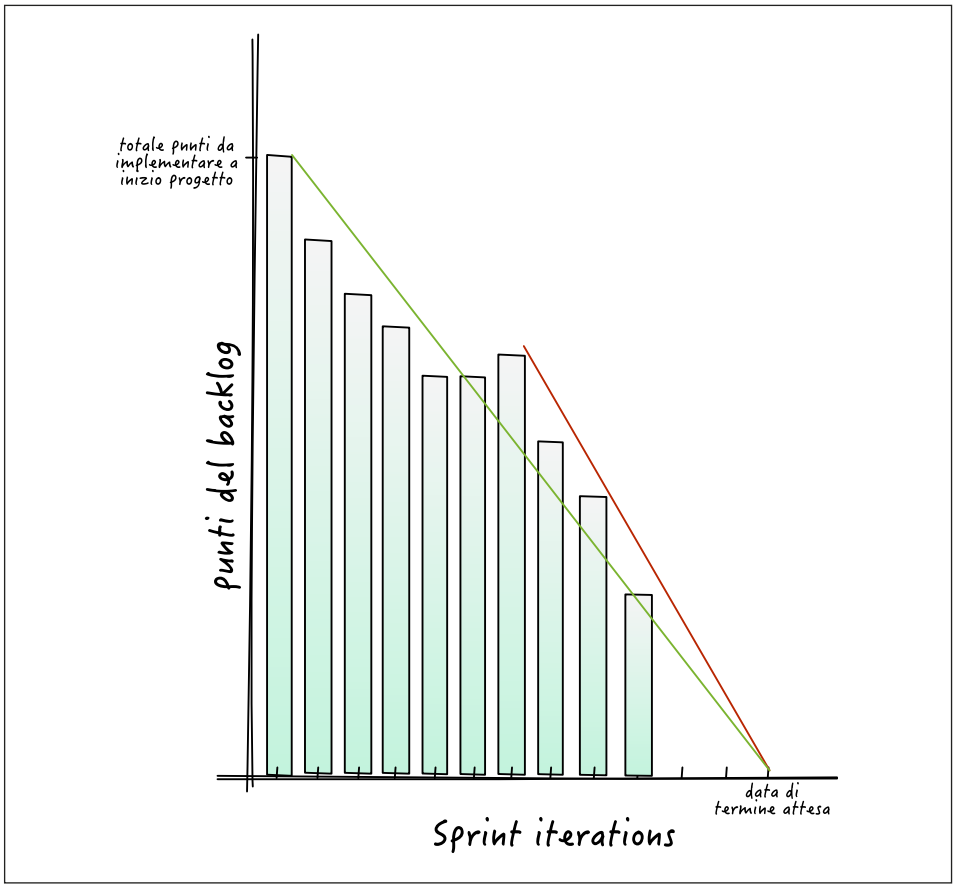


Figura 23. Se l'attuale trend non permette di rientrare nella data inizialmente stimata, una soluzione potrebbe essere quella di aumentare la velocity di sprint.

lavorare: questo la rende non produttiva all'inizio, anche se spesso con il pair programming l'effetto è meno evidente; ma si rende comunque necessario il supporto di altre persone, distraendo altre risorse dallo sviluppo.

La capacità produttiva di un gruppo, infatti, non è direttamente proporzionale alla dimensione del gruppo stesso, dato che interviene la difficoltà intrinseca derivante da far lavorare insieme le persone. All'aumentare della dimensione del team, normalmente l'aumento di produttività non è proporzionale, ma anzi spesso cresce di meno. Questo è dovuto alla complessità sociale e organizzativa di un team grande, cosa che in parte può essere indirizzata dividendo il team e usando tecniche di *scaling* per fare lo stesso lavoro, processo che comunque è inefficiente.

Insomma, per ottenere risultati positivi e restare nei vincoli di contenuto e tempo non basta avere il budget aperto e poter spendere soldi per aumentare le persone che

lavorano al progetto, o far lavorare più ore chi già fa parte del gruppo, pagando gli straordinari. Occorre essere al corrente degli intoppi che questa strategia apparentemente semplice potrebbe incontrare nel progetto.

Quanto alla soluzione "alternativa" per aumentare le performance e rimanere nei vincoli di **time** e **scope** (e in questo caso, in parte anche di **budget**) la scelta tradizionale è spesso stata quella di aumentare la **pressione sul team** stesso, tramite l'imposizione di orari di lavoro più lunghi, di una maggior pressione psicologica, l'adozione di una ferrea disciplina imposta dall'alto. Queste strategie sono in netta contrapposizione con la maggior parte dei principi agili e le prove pratiche dimostrano che le conseguenze sul medio-lungo periodo sono sempre negative (vedi [PW] e [SLK]).

Si è già abbondantemente parlato dell'importanza di lavorare secondo un ritmo sostenibile, così come della totale inutilità di regole imposte dall'alto; conviene comunque aggiungere una piccola nota sulla questione della pressione psicologica, cosa che non solo è contraria ai principi fondanti della filosofia Agile, ma si è anche dimostrata inefficace: alcuni studi [PW] dimostrano che una piccola tensione nervosa può migliorare le performance della maggior parte delle persone. Ma stiamo parlando di quel leggero stress che tiene alta la concentrazione e non di dosi maggiori di stress e "oppressione" che abbssano la qualità del lavoro prodotto in modo drastico.

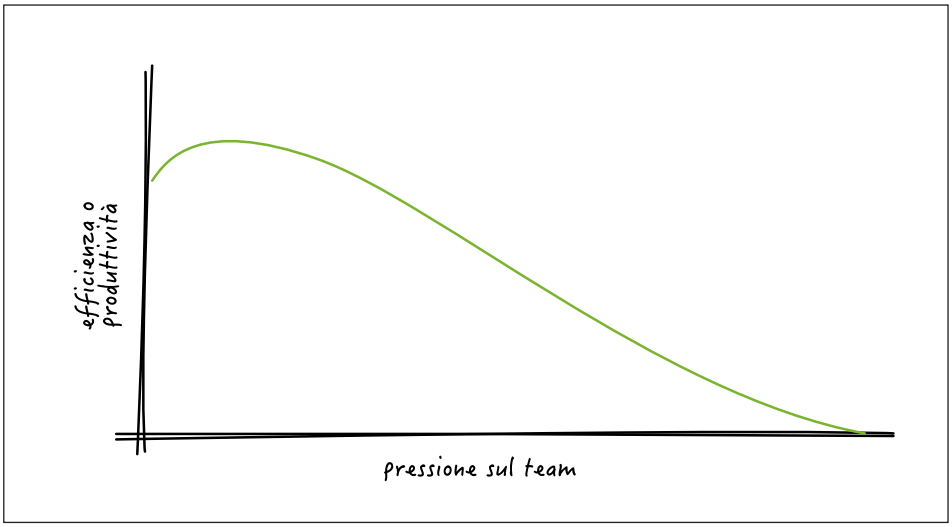


Figura 24. Andamento della qualità del lavoro svolto in relazione alla pressione del sistema e dello stress sulle persone: a volte aumentando di un poco la pressione sul team le persone lavorano in modo più concentrato e quindi producono meglio. Continuando ad aumentare la pressione, però, la produttività scende rapidamente verso lo zero. L'andamento della curva, così come il picco massimo sono caratteristici di ogni persona. In alcuni casi la curva scende immediatamente senza portare alcun miglioramento.

Fixed scope

In questo caso il **contenuto** del **progetto** è talmente importante da prevalere sugli altri vincoli: il **tempo** viene lasciato **libero** e di conseguenza anche il **budget**; in caso di ritardo sui tempi di consegna per esempio, il progetto viene prorogato fintanto che non si completano tutte le funzionalità previste in fase di pianificazione.

Questa configurazione offre una elevata libertà operativa, ma **raramente** viene applicata, dato che non consente di prevedere il costo complessivo di progetto. Inoltre impatta in modo non banale su tutti gli stakeholder di progetto: nel progetto a **fixed scope** chi deve installare il prodotto, chi deve venderlo, chi deve gestire il rapporto commerciale con il cliente trova grande difficoltà per potersi organizzare. Per questo spesso l'ipotesi **fixed scope** non viene accettata con piacere.

Fixed date

In questo caso si **fissa** la **data** di **consegna** e quindi si può contare su una certa flessibilità sullo scope finale. Per certi versi questo tipo di organizzazione progettuale è quella che si avvicina maggiormente alla filosofia di Scrum e delle metodologie agili.

Fissare la **data** e lasciare **libero** lo **scope** vuol dire infatti che, in caso di rallentamenti sulla lavorazione delle varie parti del progetto, pur mantenendo fissa la data di consegna, si potrà concordare con il cliente su quali funzionalità potranno essere escluse dal progetto o rimandate a una release successiva.

Dato che il **Product Backlog** è ordinato sempre in modo descrescente rispetto al valore o all'importanza, in ogni step di progetto quello che rimane da completare è certamente meno importante rispetto a quello che è già stato realizzato.

Come si gestisce nella pratica un progetto agile con vincoli?

Dopo che si sono visti i vari casi in cui un progetto può ricadere, vediamo adesso qual è l'approccio che si può perseguire in un progetto condotto con metodologia agile.

In linea di principio un progetto Scrum finisce quando sono implementate tutte le funzionalità; il team organizza il proprio lavoro pianificando le implementazioni delle storie per sprint.

Normalmente il **PO** vuol avere un'indicazione di massima della data di consegna del progetto; in tal caso, si prova a fare una stima approssimativa delle cose da fare: in pratica si realizza una prima definizione grossolana del **Product Backlog**. In questa fase non sempre è necessario stimare l'intero progetto, ossia non necessariamente si deve valutare tutto il Product Backlog, ma può essere utile fare una previsione delle delivery fondamentali.

Se serve, o se è richiesto si può provare a fornire una pianificazione più precisa, lavorando come al punto precedente, ma andando più in dettaglio. In questo caso può aver senso, specialmente nei progetti più grandi, fare una pianificazione a livello di roadmap e di release in modo da avere un livello di granularità adeguata sia sul lungo che sul breve periodo.

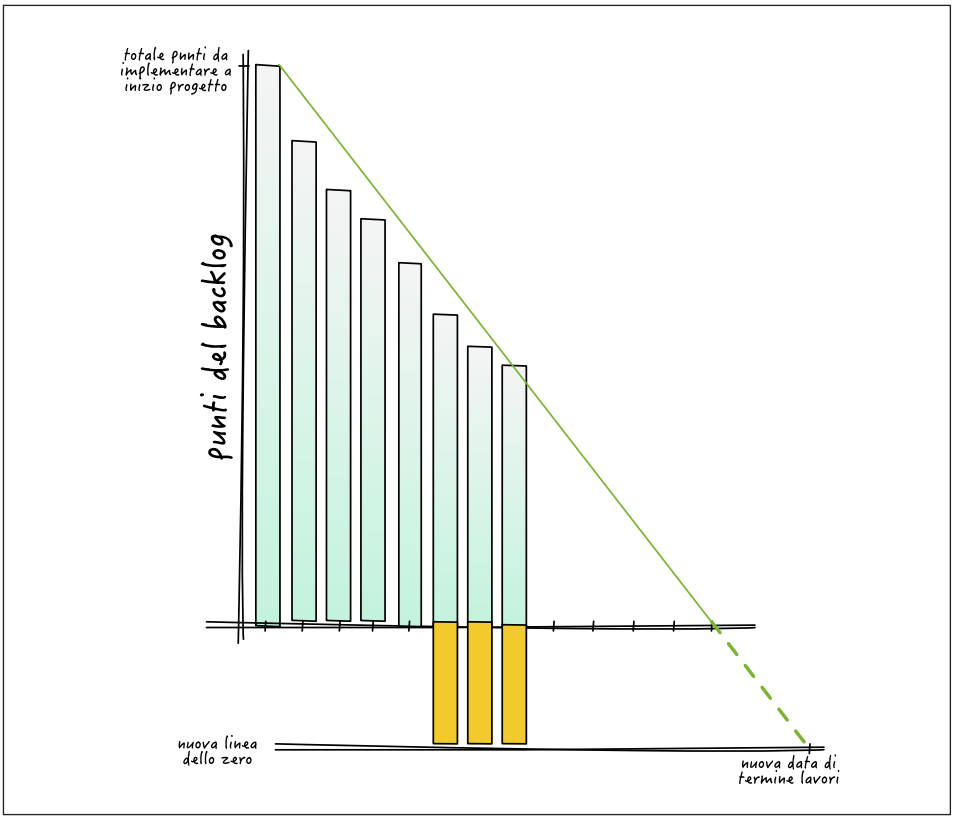


Figura 25. Il cliente, nel mezzo del progetto, chiede di aggiungere nuove funzionalità al prodotto finale. In questo caso spesso le nuove funzioni si aggiungono sotto la linea delle ascisse e si punta a considerare terminati i lavori quando si arriverà alla nuova linea dello zero.

Il risultato fin qui ottenuto è pur sempre basato su stime che possono subire variazioni durante il progetto (cambio di requisiti, maggior dettaglio delle informazioni raccolte, etc.). Si possono quindi usare due tecniche per garantire che quanto pianificato si avveri: buffer sulle stime (**schedule buffer**) o lavoro organizzato per lotti temporali definiti (**time boxing**) tagliando lo scope se non si riesce ad arrivare a quanto desiderato (**feature buffer**).

Se il progetto è **fixed budget** si può pianificare il backlog e calcolarne il costo. Se la stima è affidabile, usando per esempio i progetti passati come riferimento, si potrà avere una misura attendibile e quindi si dovrebbe poter rispettare il budget prefissato. Analogamente, se il progetto è **fixed date**, si opera nello stesso modo, ponendo come vincolo il **tempo** invece che il budget: di fatto le due grandezze sono strettamente collegate.

Il problema dei due casi precedenti è che spesso il cliente (o il management) tende a forzare il fornitore con date e costi surreali, complicando non di poco la riuscita

del progetto. Detto ciò, se si lavora in ordine di priorità, le cose che non si riusciranno a completare a causa di una stima non precisa (tutte le stime sono imprecise), saranno sempre quelle a più bassa priorità e quindi più facilmente negoziabili o rimandabili ad una release successiva.

Gestione delle stime in pratica

Indipendentemente dal tipo di configurazione e di vincoli che saranno definiti sul progetto, qualora si voglia avere un'idea di **quando** finirà il progetto, del **costo** o del **numero di cose** che si potranno consegnare prima di una determinata data di consegna, la prima cosa da fare è provare a valutare l'**effort** necessario per la realizzazione di **tutte** le **funzionalità** richieste.

Durante le prime fasi di progetto, il backlog sarà composto per la maggior parte da storie grandi (features o più probabilmente epiche), cosa che non semplifica questa attività: la stima in punti in questo caso è poco efficace: i pezzi sono troppo grandi per poter consentire di stabilire un punteggio che abbia un qualche senso.

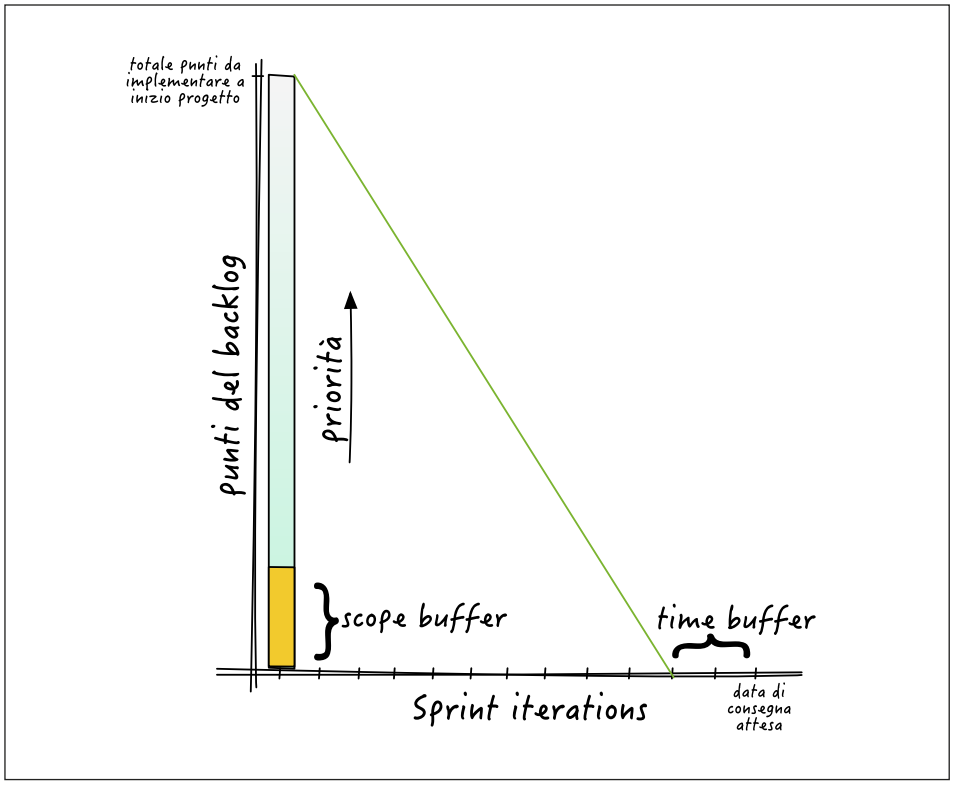


Figura 26. La gestione della release di un prodotto agile passa per la gestione di aree di variabilità sia sulle tempistiche (time buffer) che sulle funzionalità di implementare (scope buffer).

Lo stesso Planning Poker non è applicabile: non è un caso che nelle carte del planning poker il punteggio più alto sia spesso 100 e non di più; e infatti spesso il team utilizza tecniche alternative come assegnare alle epiche etichette che ne esprimano macroscopicamente la dimensione, per esempio usando le taglie delle magliette (XL, L, M, S, XS...). Dato che non è possibile sommare le lettere delle taglie per dedurre l'effort complessivo del progetto — sommando una epica XL con una S cosa si ottiene? — il problema resta irrisolto: come si arriva a una stima "maneggiabile" dell'effort complessivo del progetto? E vedremo tra poco cosa vuol dire "maneggiabile".

In questi casi una strada percorribile può essere procedere per raffinamenti del backlog, smontando e dettagliando, in modo da arrivare a un elenco di elementi più piccoli; in questo caso si procede fino a ottenere "pezzi" per i quali il team si sente confidente nell'esprimere una stima in punti. Per esempio, sempre rifacendosi alla storia campione si potrebbe chiedere al team se un elemento è grande 5, 10 o 20 volte la storia campione.

Lo scopo di questa attività è arrivare a poter esprimere un numero dato dalla somma di tutti questi "pezzi" medio grossi, ossia ricavare il **punteggio complessivo P** del **Product Backlog**. Questo numero rappresenta, con un ragionevole tasso di approssimazione, l'effort complessivo necessario per implementare tutte le funzionalità che il cliente ha chiesto.

Come è facile comprendere si tratta di una stima estremamente **approssimativa** e difatti, differentemente da quello che si fa spesso in questi casi, tale stima verrà poi rivista e raffinata in seguito, tramite nuove sessioni di votazione sugli elementi del backlog (che nel frattempo saranno stati ulteriormente smontati grazie al **Refinement**).

Se è vero che il valore di **P**, ossia il punteggio delle cose da realizzare, diminuisce fisiologicamente iterazione dopo iterazione (il team implementa le storie e quindi toglie punti dal backlog), dall'altro potrebbe aumentare o restare costante perché le nuove informazioni potrebbero aiutare a ricalcolare le stime in modo più preciso.

Supponendo di conoscere quindi la **sprint velocity V** del team di sviluppo, si potrà calcolare il numero **N** di sprint necessari per realizzare il progetto, tramite il calcolo:

```
N = P / V
```

Essendo poi **G** il numero di giorni di cui si compone uno sprint, allora si potrà calcolare il tempo in giorni necessario per sviluppare il progetto tramite la seguente moltiplicazione:

```
T = N * G
```

A seconda della precisione del valore di **P** e di **V** si potrà ottenere una risposta più o meno attendibile. A inizio progetto il valore ottenuto sarà estremamente poco realistico. Essendo Scrum un processo **adattivo**, il team impara a raffinare, iterazione dopo iterazione sia il valore di **V**, perché il team conosce meglio le proprie potenzialità, sia di **P**,

perché il team spesso ripete la misurazione del peso complessivo backlog, che dopo le numerose attività di raffinamento, finisce per essere molto preciso.

Già dopo poche iterazioni, il valore di **T** risulta essere più attendibile e il team è in grado di capire se le ipotesi fatte inizialmente potranno essere rispettate.

Un aspetto importante, quando si parla di stime, è legato al **commitment**, ossia all'impegno che il team si prende per lo sviluppo delle storie: in Scrum il team di sviluppo esegue una previsione sul lavoro che potrà essere eseguito all'interno dell'iterazione in fase di **Sprint Planning**, ma non fa nessuna promessa sull'intero progetto. Il **commitment** verso gli **stakeholder** è, in realtà, del **PO**, il quale, visto l'output del team, "scommette", per così dire, su una certa data di consegna, spesso gestendo **schedule** e **feature buffer**.

Per concludere: stime, stime agili, scope management e descoping

Per quanto visto fino a questo punto, appare evidente che anche in Agilità si possa parlare di stime, si possa provare a prevedere quando terminerà un progetto o quanto potrà costare. Anche in Scrum si può stimare tutto un **Product Backlog** e calcolare in base alla **Velocity** la data di consegna, aiutandosi con il **Burn-Down Chart** per tenere sotto controllo lo stato di avanzamento

In Agilità si pone però molta attenzione all'importanza di fare **scope management**, (gestione dei contenuti da realizzare) cosa che per esempio si traduce nel concordare con il cliente la priorità delle cose da rilasciare. Prioritarizzare l'implementazione delle funzionalità, in caso di ritardo nelle tempistiche di consegna, semplifica molto il **descoping** (cosa può essere tolto dal progetto), perché la parte importante è già stata sviluppata, testata e validata dal cliente/utente.

Anche nel project management tradizionale fare descoping è formalmente possibile, ma in realtà viene fatto raramente e solo in presenza di pressioni estreme. Tipicamente si ragiona con la regola "non si può togliere nulla"; dopodiché, quando il tempo stringe e l'ultima feature importante è stata realizzata, centinaia di cose minori spariscono nell'arco di una notte. Quindi, perché non far diventare questa pratica una regola di progetto?

Se si accetta la possibilità di fare **scope management**, il valore della stima di tutto il **Product Backlog** diminuisce molto perché lo **scope management** va a influire proprio su quel valore: dov'è il limite del progetto? Da variabile immutabile universale, diventa un concetto molto fluido.

Per questo in Agilità lo **scope** spesso non viene visto come uno dei parametri fondamentali nella gestione del progetto; lo sono il **time**, il **budget**, ma lo **scope** viene messo in secondo piano in favore dei **business objectives**. Lo **scope è un dettaglio tecnico**, soggetto a cambiare, con cui si possono ottenere gli obiettivi di business. Anche se, spesso, le due cose sono confuse, volontariamente o meno.

Riferimenti

[DM] E. Yourdon, *Death march*. Prentice Hall, 3a ed, 2015

[DMWiki] La pagina Wikipedia sul concetto di "marcia della morte" nel project management
http://en.wikipedia.org/wiki/Death_march_(project_management)

[NEM] Il movimento "No Estimates"
http://ronjeffries.com/xprog/articles/the-noestimates-movement/

[MF] Martin Fowler, "PurposeOfEstimation"
http://martinfowler.com/bliki/PurposeOfEstimation.html

[PW] T. DeMarco, T. Lister, *Peopleware: productive projects and teams*. Addison-Wesley Professional, 3a ed., 2013

[SLK] T. DeMarco, *Slack: getting past burnout, busywork, and the myth of total efficiency*. Crown Business, 2002