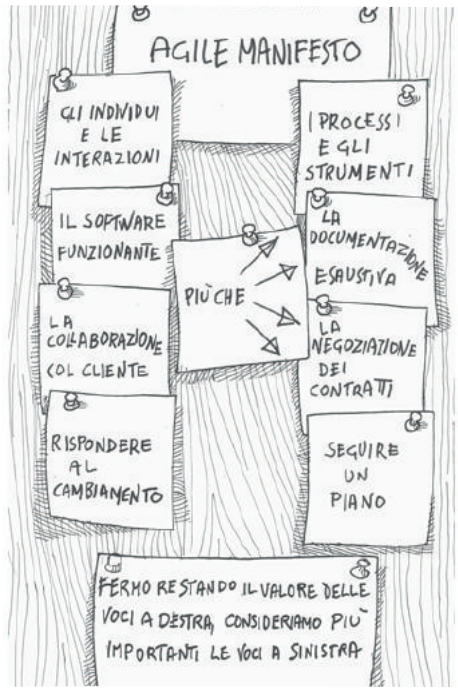
Stefano Leli
Giovanni Puliti
Giulio Roggero

GUIDA GALATTICA PER AGILISTI

INTRODUZIONE

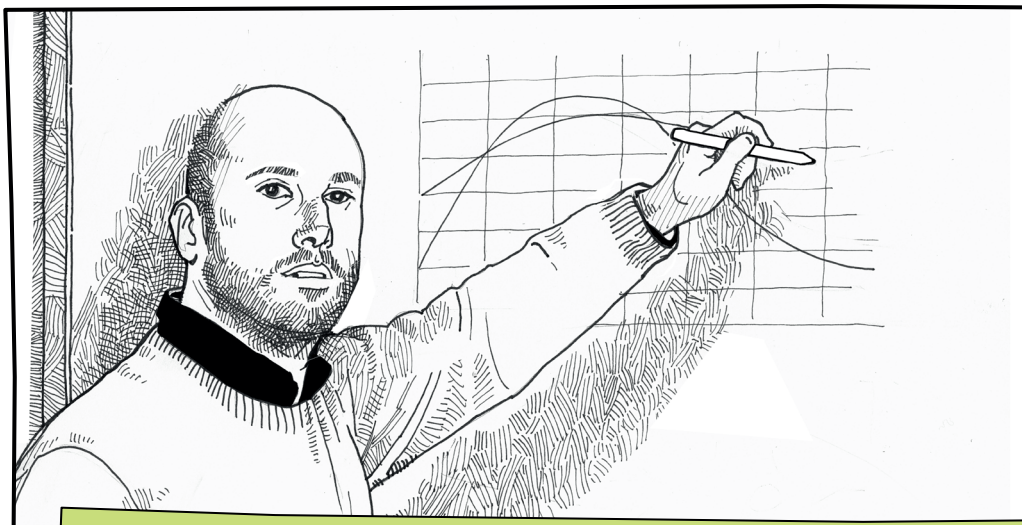
Introduzione

La Guida Galattica per Agilisti è un libro, scritto dagli articolisti di MokaByte, che parla di metodologie e di pratiche agili di gestione di lavoro. Nelle varie parti del libro, oltre a parlare di Scrum, Kanban, di principi e valori, ripercorrendo l'evoluzione del management nel XX secolo, racconteremo il percorso che ha portato alla nascita del movimento Agile.



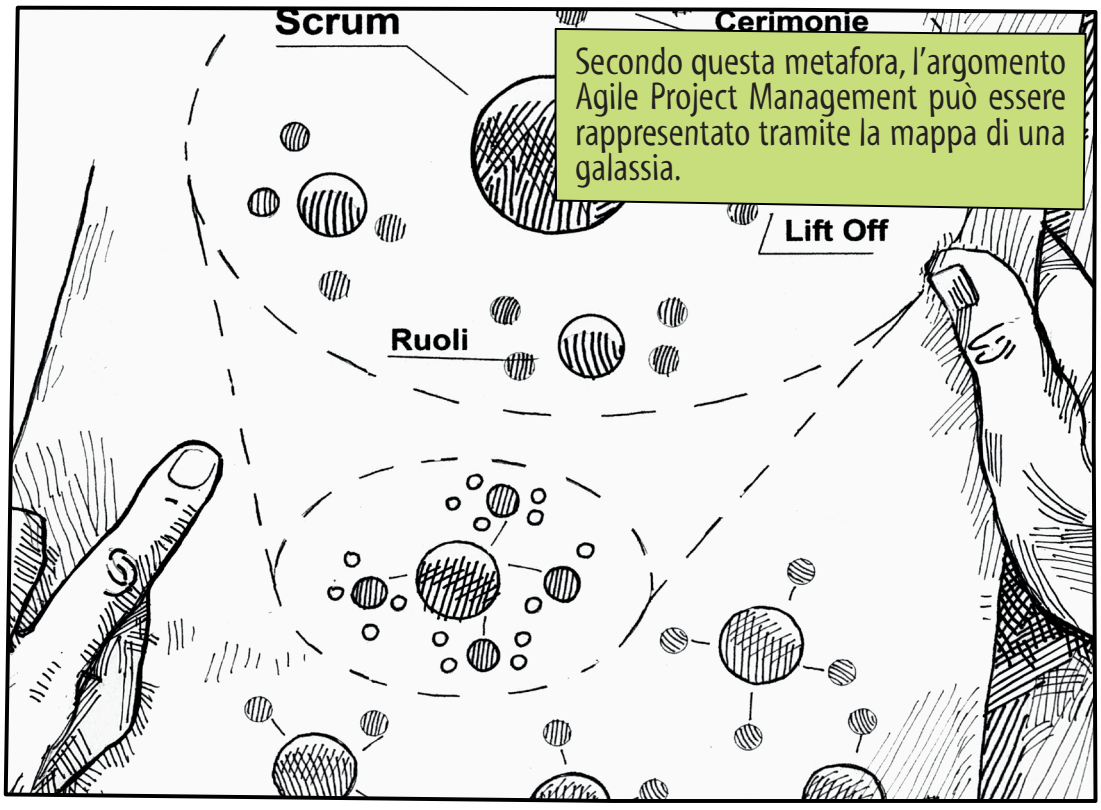
I 12 principi dell'Agile Software Development

1. Massima soddisfazione del cliente con rilasci frequenti
2. Il cambiamento è benvenuto
3. Delivery frequenti
4. Il business lavora col team
5. Fondare il progetto su persone motivate
6. Face to Face
7. Misurare il progresso con il software rilasciato
8. Ritmo sostenibile e continuativo
9. Massimizzare il numero di cose non fatte
10. Architetture e pattern emergono
11. Retrospettiva
12. Il team si auto organizza

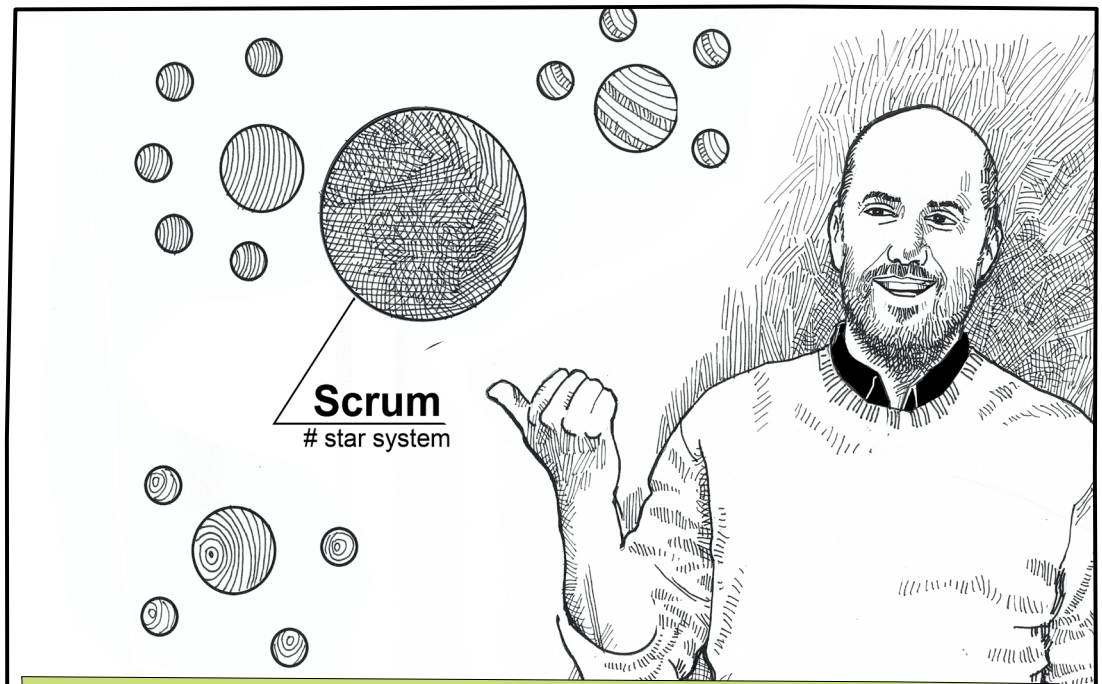


Il libro prende spunto da un'idea venuta a uno degli autori, Giovanni Puliti, il quale, nei suoi corsi, utilizza spesso la metafora della mappa spaziale per visualizzare il susseguirsi dei vari argomenti trattati in aula.

L'idea si è dimostrata utile per spiegare i concetti e permettere ai partecipanti del corso di avere sempre chiaro dove un qualche concetto si pone rispetto al resto della trattazione.

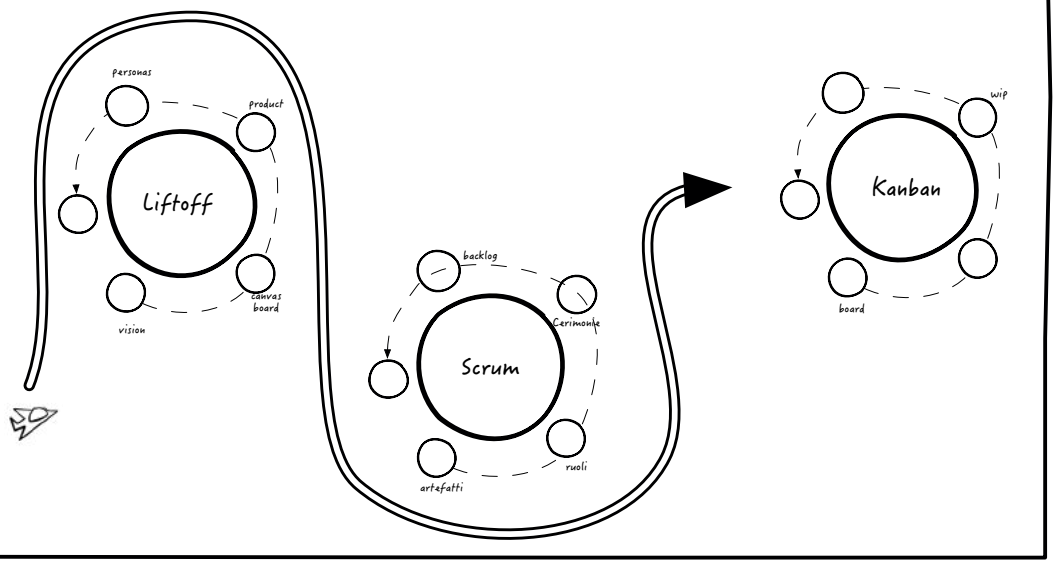


Secondo questa metafora, l'argomento Agile Project Management può essere rappresentato tramite la mappa di una galassia.

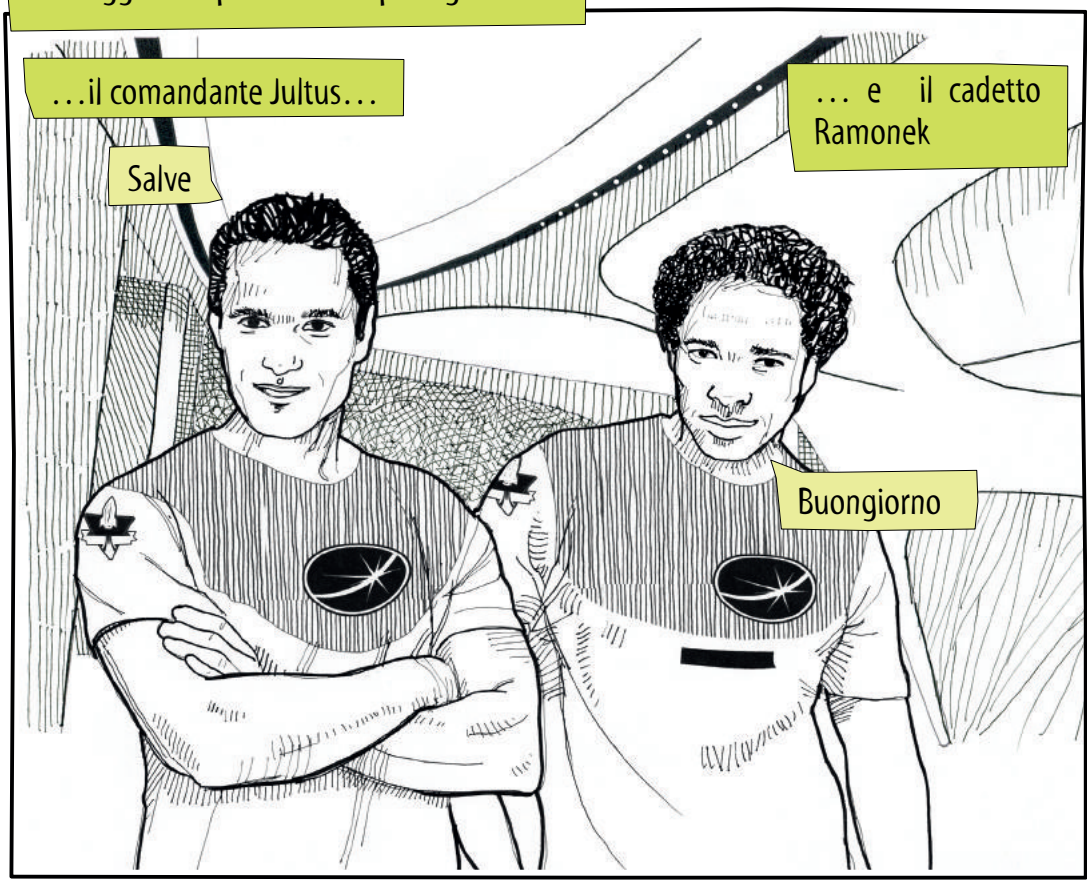


Si è deciso quindi di adottare la metafora della mappa anche in questo libro: quindi ogni sistema stellare corrisponde a una parte, ogni pianeta a un capitolo, mentre i satelliti sono i vari paragrafi dei capitoli.

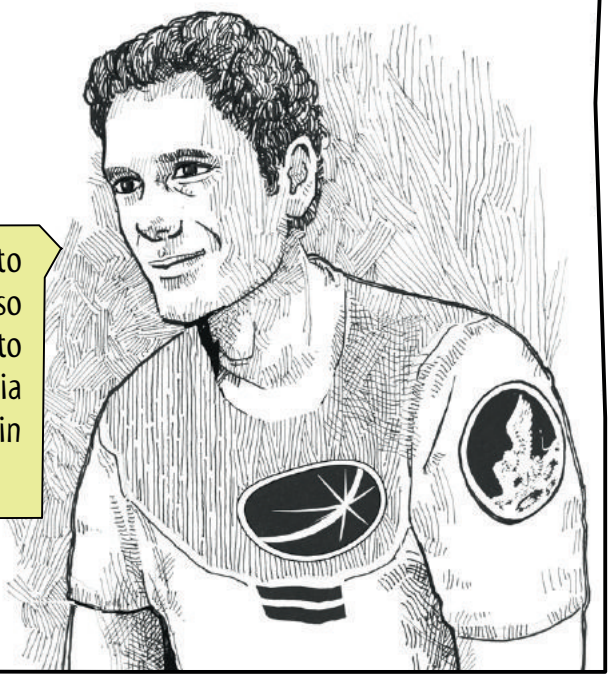
Dalla mappa al viaggio il passo è breve: per impostare un filo logico che fosse facile da seguire e tenere a mente, abbiamo pensato a un viaggio interstellare.



Un viaggio compiuto da due protagonisti…



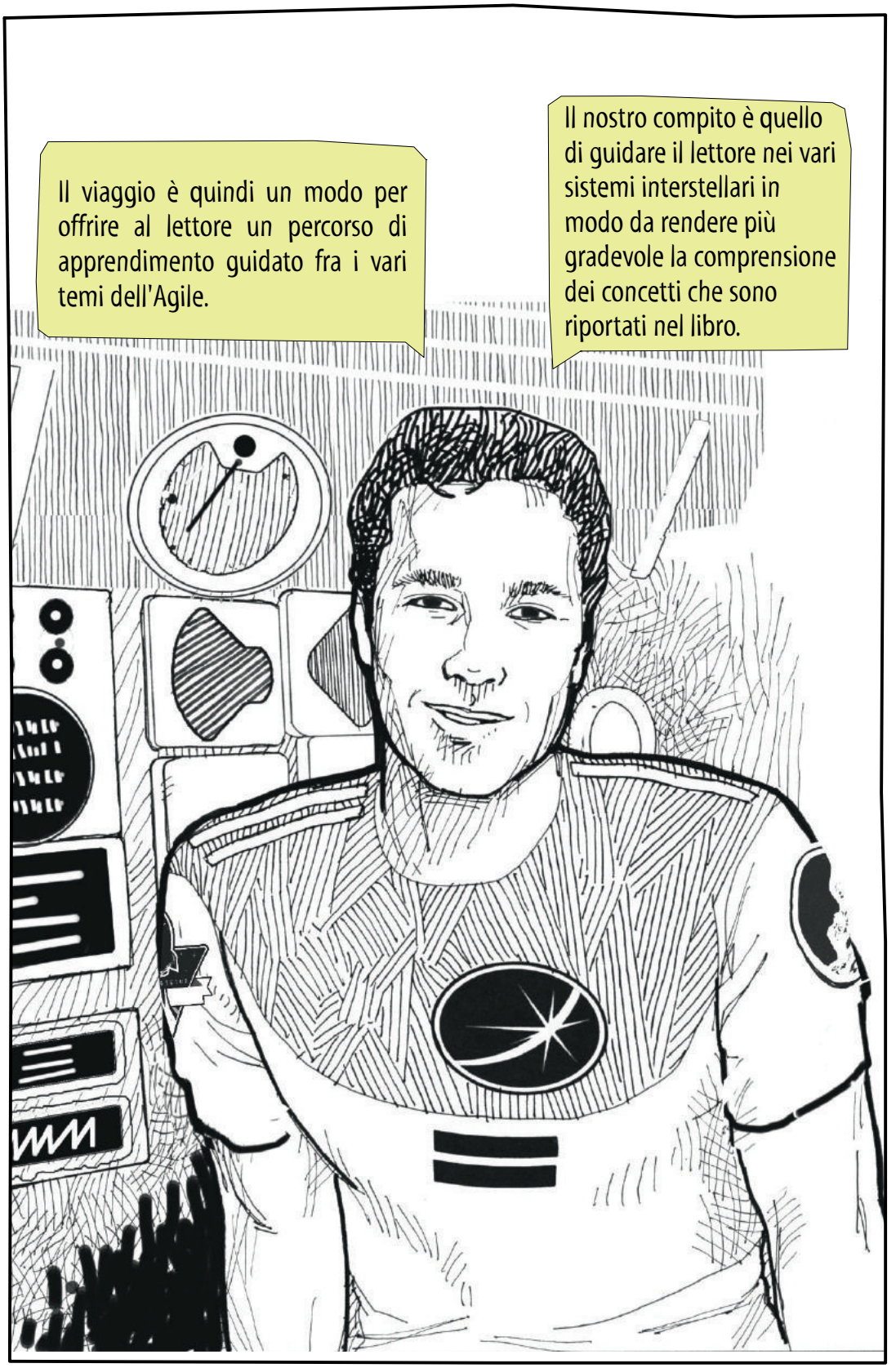
Comandante, la prego, potrebbe raccontare ai nostri lettori il senso del viaggio che intraprenderete?



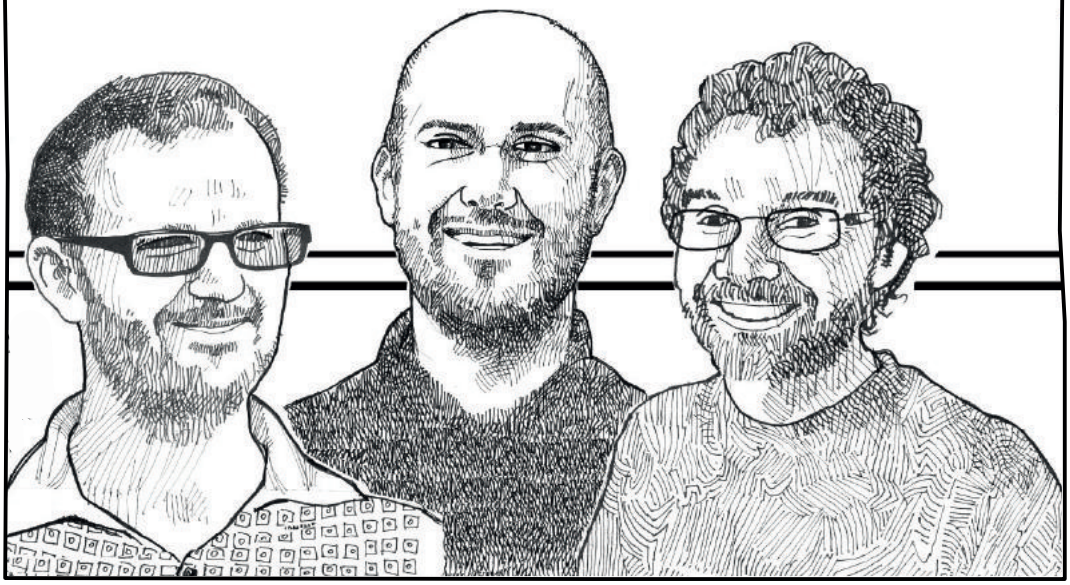
Be', il viaggio è cominciato quando, al termine del suo corso in accademia spaziale, il Cadetto Ramonek, ha richiesto la mia disponibilità a essere il suo tutor in questa esplorazione.

Sì, al termine dei miei studi, mi fu offerta la possibilità di fare questo viaggio per vedere da vicino i vari sistemi dell'Agilità.

Accettai immediatamente la proposta, perché sapevo che questo viaggio sarebbe stata un'esperienza importante per comprendere concretamente i principi dell'Agile.


Il viaggio è quindi un modo per offrire al lettore un percorso di apprendimento guidato fra i vari temi dell'Agile.
Il nostro compito è quello di guidare il lettore nei vari sistemi interstellari in modo da rendere più gradevole la comprensione dei concetti che sono riportati nel libro.

Ma questo libro si è potuto realizzare sopratutto grazie al contributo degli autori, i quali hanno condensato nei vari capitoli le loro conoscenze e l'esperienza maturata sul campo come coach, formatori e consulenti.



Giulio Roggero

ciao

Giulio è Agile Coach e Platform Designer. Aiuta le aziende — sia ICT che del mondo industriale, editoriale, telco e finance — a creare nuovi prodotti e a semplificare i loro processi, spostando l'attenzione verso il valore per il cliente finale.

Stefano Leli

ciao

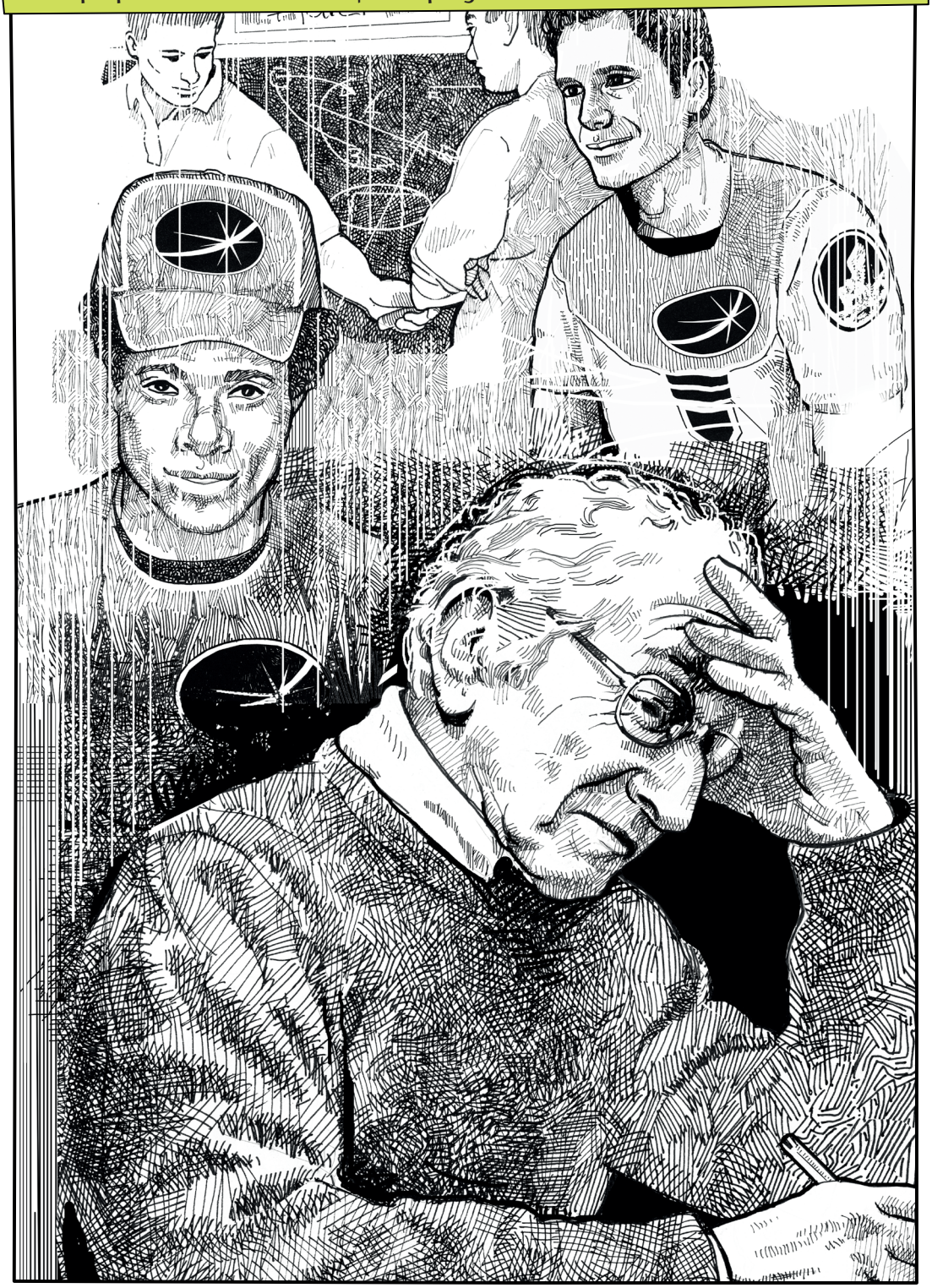
Stefano Leli è appassionato di sviluppo software fin da bambino e lavora nel settore IT da oltre 15 anni. Nel 2003 sente parlare del Manifesto Agile e inizia titubante a praticare XP. Da allora il suo approccio allo sviluppo non è stato più lo stesso e non ha più abbandonato l'adozione delle metodologie agili. È fondatore e coordinatore dell'Extreme Programming User Group Marche.
Aiuta le aziende IT a semplificare i propri processi grazie all'introduzione dei principi e valori agili.

E infine Giovanni Puliti

ciao a tutti

Giovanni è nel settore dell'IT da oltre 20 anni. Dopo aver lavorato a lungo come esperto di tecnologie e architetture, è passato a erogare consulenze sul project management. Con alcuni collaboratori nel 1996 crea MokaByte, la prima rivista online italiana dedicata a Java. È autore di numerosi articoli e libri.

Per ultimo ma non per questo meno importante, vi presentiamo l'autore dei fumetti che accompagnano questo libro.
Giampiero, il babbo di Giovanni, da anni disegna e dipinge. Quando gli è stato proposto di contribuire a questo progetto... ha accettato con entusiasmo.



Struttura, temi, autori del libro

Come nasce questa guida

Guida galattica per agilisti è un libro che esplora la "galassia" delle metodologie agili. Nasce dalla revisione e dall'aggiornamento di articoli pubblicati su **MokaByte** nel corso degli ultimi anni, e presenta al lettore svariate tematiche inerenti il mondo Agile da diversi punti di vista.

A fronte di un ricco panorama di titoli in inglese su questi argomenti, abbiamo voluto realizzare un testo in italiano che non fosse la mera riproposizione dei best seller statunitensi sulle metodologie Agile e sulla gestione di processo Lean. Uno dei punti di forza del libro, infatti, sta nell'approccio multidisciplinare e "**olistico**" agli argomenti, che tenta di dare un'immagine a **tutto tondo** del complesso mondo legato alle discipline di gestione di processo, di project management e di metodologie di sviluppo del software.

Nelle varie parti di **Guida galattica per agilisti**, infatti, non si parla solo di **cosa** fare per gestire un processo di sviluppo del software, magari indicando bene **come**. Nel libro, invece, oltre alla **descrizione dettagliata** delle metodologie agili (**Scrum e Kanban** anzitutto), si offre sempre una visione più ampia, che abbracci tutto il contesto: dalla **visione** al **prodotto**, dalla **dinamica** dei gruppi di lavoro al **miglioramento continuo**, dai valori ai **principi** tipici della filosofia **Lean**.

In questo libro sono finite le esperienze maturate direttamente sul campo, da parte degli autori: dopo anni passati all'interno di progetti software, di coaching fatto presso organizzazioni e team, di relazioni con gruppi di sviluppo, aziende e utenti finali, abbiamo voluto condividere ciò che abbiamo sperimentato essere efficace per affrontare i problemi incontrati da noi e dai nostri interlocutori.

In questo senso il testo rappresenta una sorta di **guida** a supporto del percorso di crescita dentro la **galassia** di concetti e di informazioni che riguardano queste tematiche: da qui il titolo del libro. L'auspicio degli autori è di riuscire a condensare nel libro proprio questo **approccio globale**, nel tentativo di tenere sempre in vista il quadro d'insieme della "galassia Agile".

Infine una nota sul modo in cui è stato pensato e realizzato il libro: la **Guida galattica per Agilisti** sarà una pubblicazione in continua evoluzione. Da un lato le varie parti del libro rappresentano la piattaforma da cui attingere per la creazione di iniziative come corsi, eventi, conferenze e altro ancora. Dall'altro è un sistema "vivo" poiché pensiamo di farlo evolvere continuamente con nuovi contenuti e revisioni di quelli esistenti, anche grazie al contributo di colleghi che svolgono il nostro stesso lavoro e agli amici che fanno parte della community.

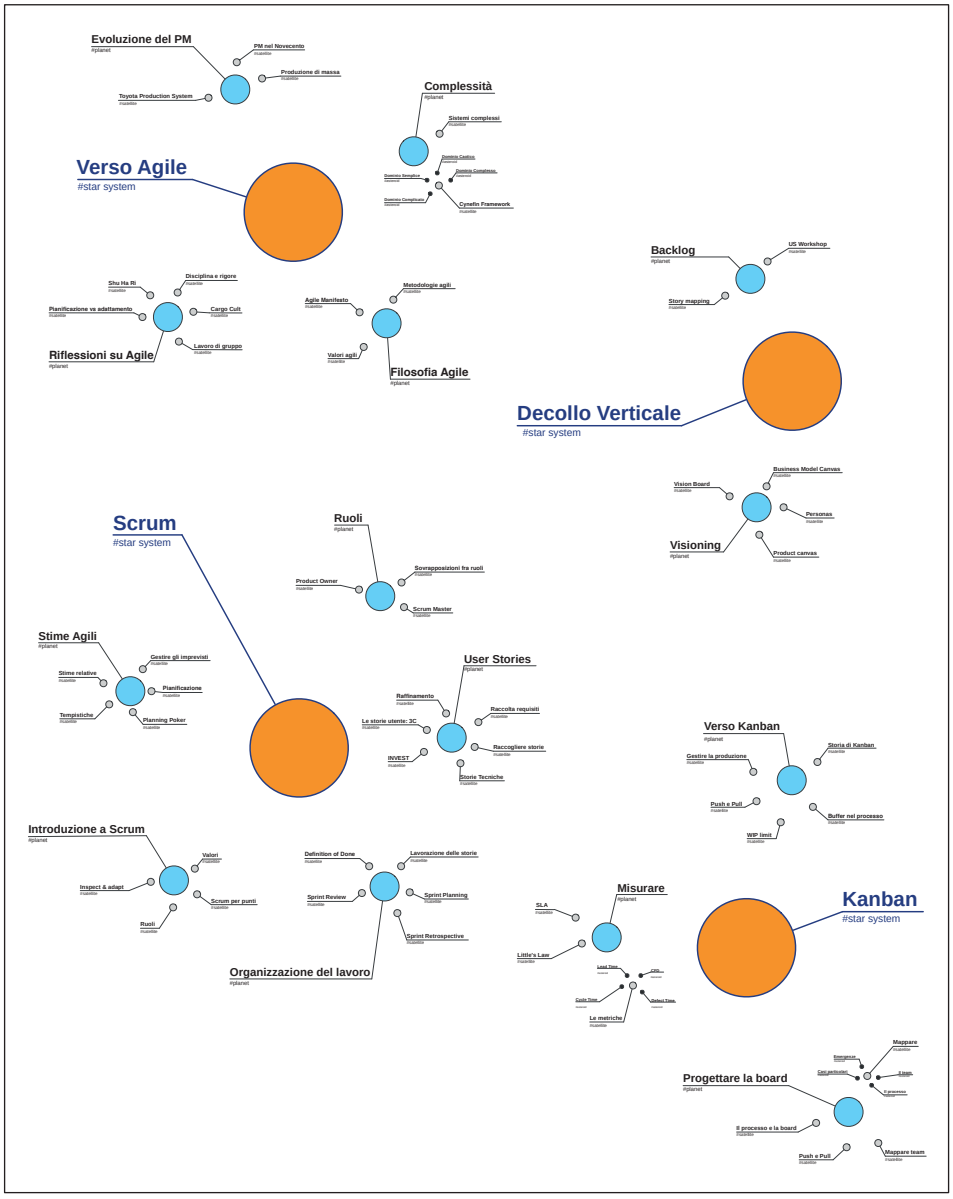
Struttura del libro

La metafora del viaggio e la mappa della galassia

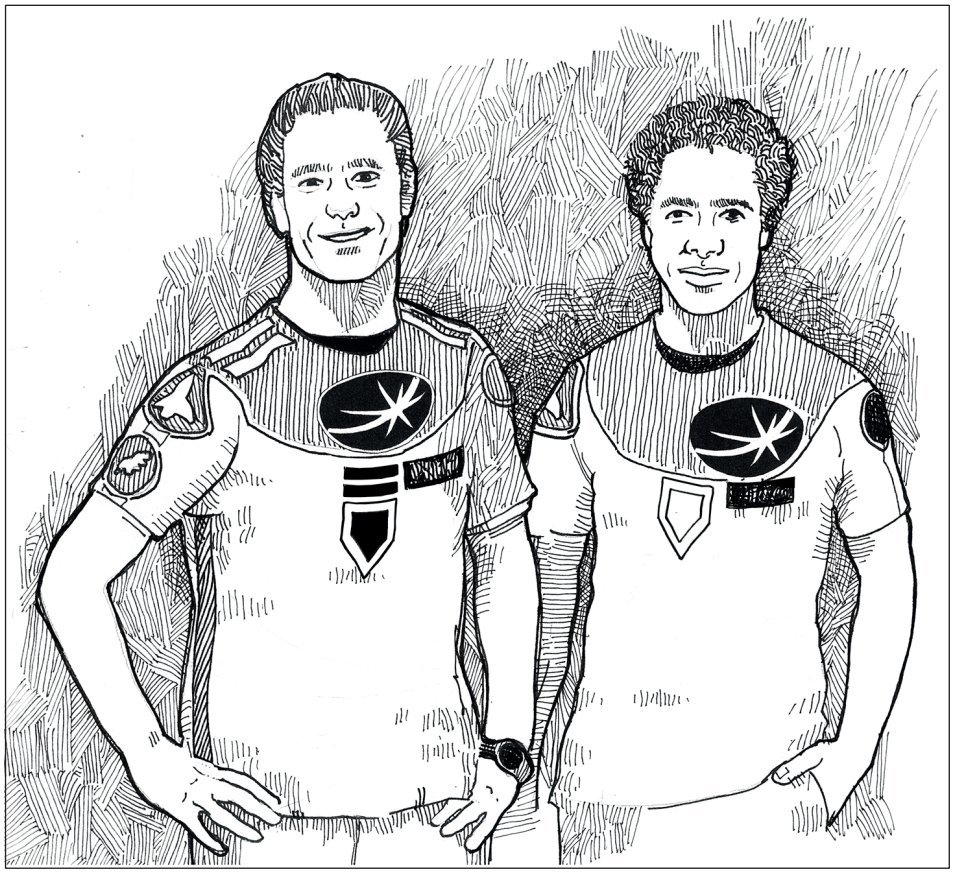
L'idea alla base del libro è di guidare il lettore a "esplorare" i vari argomenti tramite la **metafora** di un **viaggio** interstellare, condotto nella "galassia" delle metodologie agili: da questo deriva il titolo del libro, che richiama inoltre il celebre romanzo fantascientifico/umoristico di Douglas Adams.

Ogni passaggio di questa esplorazione è illustrato da una **mappa stellare** che consente al lettore di orientarsi tra i vari argomenti trattati. Pertanto, nella nostra metafora:

- ogni parte del libro rappresenta un sistema solare da esplorare;
- ogni capitolo all'interno delle parti rappresenta un pianeta da conoscere;
- i satelliti rappresentano argomenti collaterali trattati all'interno dei capitoli.



Una visione di insieme della mappa galattica, con i vari sistemi stellari.



Il comandante Jultus e il cadetto Ramonek, le due guide che accompagnano il lettore nel viaggio interstellare riportato all'interno del libro.

Gli argomenti

Gli argomenti trattati sono organizzati secondo il seguente schema:

- Sistema stellare **Verso Agile**
- Sistema stellare **Dalla visione al prodotto**
- Sistema stellare **Scrum**
- Sistema stellare **Kanban**

In una futura seconda edizione, si aggiungeranno con ogni probabilità ulteriori argomenti:

- Sistema stellare **Retrospettive agili**
- Sistema stellare **Aspetti psicologici e soft skills**
- Sistema stellare **Portofolio Management**

Il fumetto

Ad ogni parte del libro è abbinato un **fumetto** che introduce ai temi affrontati all'interno dei vari capitoli. Il fumetto, disegnato da Giampiero Puliti, racconta le avventure di due personaggi, il comandante Jultus e il cadetto Ramonek, i quali affrontano il viaggio fra i vari sistemi stellari. Questo viaggio fornirà a Ramonek l'occasione per comprendere valori, principi e pratiche delle metodologie agili.

Gli autori

Stefano Leli è appassionato di sviluppo software fin da bambino e lavora nel settore IT da quasi venti anni. Nel 2003 sente parlare del **Manifesto Agile** e inizia titubante a praticare XP. Da allora il suo approccio allo sviluppo non è stato più lo stesso e non ha più abbandonato l'adozione delle metodologie agili. È fondatore e coordinatore dell'eXtreme Programming User Group Marche. Aiuta le aziende IT a semplificare i propri processi grazie all'introduzione dei principi e dei valori agili. È socio fondatore e agile coach di **Agile Reloaded**.

Giovanni Puliti è nel settore dell'IT da oltre venti anni. Dopo aver a lungo lavorato come esperto di tecnologie e architetture all'interno di progetti web enterprise, è passato a erogare consulenze in ambito di project management. Con alcuni collaboratori, nel 1996 crea **MokaByte**, la prima rivista online italiana dedicata a Java; è autore di numerosi articoli e di libri. È socio fondatore e agile coach di **Agile Reloaded**.

Giulio Roggero, *platform designer* e imprenditore, nel 2006 ha iniziato a impiegare con successo pratiche agili nei suoi progetti coinvolgendo i colleghi e i clienti. Nel 2013 è stato promotore dell'iniziativa di Agile Reloaded di cui ora è *advisor*. È fondatore e CTO di **Mia-Platform** ed è partner di **Intré Srl**. Aiuta le aziende, sia ICT che del mondo industriale, editoriale, telco e finance, a creare nuovi prodotti e a semplificare i loro processi, spostando l'attenzione verso il valore per il cliente finale.

Le parti del libro

Parte 1 – "Verso Agile"

di Giovanni Puliti

Perché è così importante parlare di **Agile**? Qual è il rapporto tra metodologie agili e processi di produzione "tradizionali"? Nei capitoli di questa prima parte ripercorriamo quanto accaduto nel "**secolo breve**" attraverso il racconto dei fatti, dei protagonisti, delle idee, dei successi e dei fallimenti nella gestione di progetto e di processo del mondo industriale e software; cercheremo poi di evidenziare il percorso che ha dato vita a un certo modo di intendere la gestione di progetto.

Vedremo pertanto come e perché, verso la fine del Novecento, sia stato messo in discussione il **project management classico** (quello del **micromanagement**, della **scomposizione** delle attività e dei processi **deterministici**); cercheremo di capire da cosa sia nata la **filosofia Lean** e come si sia sviluppato il **movimento Agile**.

Questi gli argomenti trattati nella parte 1:

- l'evoluzione del project management: dalla produzione di massa al **Lean**;
- la **complessità** dei sistemi e il modello **Cynefin**;
- l'approccio agile agli **scenari complessi**;
- concetti di **Agile Management.**

Parte 2 – "Dalla visione al prodotto"

di Stefano Leli, Giulio Roggero, Giovanni Puliti

Qual è il processo per trasformare un'**idea** in un **prodotto**? Come si possono identificare in modo efficace i **bisogni** di un utente in modo che il prodotto possa soddisfarne le esigenze? Ma prima ancora, chi è l'**utente** del prodotto?

In questa seconda parte si delinea il processo che permette di formalizzare la **visione** d'**insieme** del prodotto, di definire i **requisiti fondamentali** e impostare il **progetto** in modo che sia compatibile con un approccio iterativo e incrementale, tipico delle metodologie agili.

Questi gli argomenti trattati nella parte 2:

- la definizione della **vision**: dall'**Elevator Pitch** alla **Vision Board**;
- identificare le caratteristiche del prodotto: **Product Canvas**;
- come pianificare l'organizzazione del lavoro: dall'idea al **Product Backlog**; raccogliere le storie con la tecnica dello **User Story Mapping**;
- definire il **modello** di **business**;
- come **coordinare** tutti gli strumenti di lavoro per definire il processo di trasformazione dall'idea al prodotto.

Parte 3 – "Scrum"

di Giovanni Puliti

Ruoli, **artefatti** e **cerimonie** sono le tre colonne portanti di **Scrum**: in questa terza parte il lettore esplorerà i principi e le pratiche della metodologia Scrum.

Questi gli argomenti trattati nella parte 3:

- introduzione al framework: una panoramica delle **parti essenziali** di Scrum;
- l'organizzazione: la **gestione iterativa** e **incrementale** del lavoro, il **Product Backlog**, le "**cerimonie**";
- le **storie utente**: raccogliere i requisiti, esplicitare il valore e le priorità;
- i **ruoli**: compiti e responsabilità di **Product Owner**, **Scrum Master** e **Team** di **sviluppo**;
- le **stime agili**: dalla pianificazione a controllo con l'uso del **burndown chart.**

Parte 4 – "Kanban"

di Giovanni Puliti

Kanban è uno **strumento** di lavoro che nasce in Giappone nell'ambito della manifattura **Lean**, come metodo per il controllo dei **flussi** di **produzione**. Nella sua versione riveduta e adattata, **Kanban** ben si integra con altre metodologie agili e infatti, a partire

dal 2007 Kanban è stato introdotto nel **software management** come metodologia di gestione e controllo dello sviluppo del software

Nei capitoli di questa quarta parte, tramite molti esempi e immagini, verranno affrontati concetti fondamentali per imparare la metodologia e applicarla con successo alla gestione di un progetto software.

Questi gli argomenti trattati nella parte 4:

- introduzione a Kanban: **storia, concetti fondamentali**;
- come è fatta una **board** e come è fatta una **card**;
- come **mappare** un **processo** per progettare una **Kanban Board**;
- **ottimizzazioni, WIP limit** e **buffer**: concetti avanzati presi in prestito dalla teoria delle code e dai fondamentali della Lean Production.

Ringraziamenti

Da "curatore" oltre che autore di questo libro, desidero aggiungere una **nota** e dei **ringraziamenti**.

La scrittura di *Guida Galattica per Agilisti* è iniziata ormai svariati anni fa. Stavo progettando un corso introduttivo alle tematiche Agile e realizzai alcuni diagrammi per facilitare la presentazione degli argomenti. Questi schemi cominciarono sempre più ad assomigliare a una piccola "galassia", fatta di stelle e pianeti che vi ruotano intorno: ogni stella era una parte, i pianeti i capitoli, i satelliti i paragrafi. Avevo trovato una bella metafora — almeno a parer mio — con un diagramma a metà fra una *mindmap* e un'infografica.

Con alcuni altri autori, cominciammo a scrivere su MokaByte i primi articoli che poi aumentarono diventando la base per i vari capitoli del libro. Ma si è trattato di un processo lungo e impegnativo, che ha richiesto un lavoro di aggiornamento e revisione portato avanti in alcuni anni.

Per il titolo cercammo qualcosa che ricordasse l'idea del viaggio nello spazio: *Guida Galattica per Agilisti* fu scelto non come "plagio" ma come riconoscimento al grande libro di Douglas Adams.

L'idea di aggiungere il **fumetto** "riassuntivo" all'inizio di ogni parte fu accolta con piacere da tutti. Mio padre — o meglio, mio **babbo**, come si dice dalle mie parti — fu "assunto a costo zero" per la preparazione dei fumetti. A lui va il primo ringraziamento, per il lavoro svolto e per la pazienza dimostrata nell'accogliere tutte le mie indicazioni. Diversi lettori ci hanno più volte fatto i complimenti per la scelta: "Siete riusciti a trasmettere con un fumetto questi concetti, probabilmente per via di una buona padronanza dell'argomento". Probabilmente è vero. Mi sembra giusto girare questi apprezzamenti al disegnatore che, a sua volt,a ha saputo tradurre in forma visiva i concetti che di volta in volta gli passavamo.

Vorrei poi ringraziare i compagni di avventura — e altri autori di questo libro — **Giulio** e **Stefano**, per la fiducia e per il contributo nella scrittura: Giulio fu uno dei primi a tirarmi dentro all'agilità, Stefano è stato instancabile, e prezioso, critico di quello che ogni volta gli proponevamo.

Grazie a **Claudio Bergamini**, presidente di Imola Informatica S.P.A., di cui MokaByte fa parte, per il supporto materiale e spirituale fornito in questi anni, animato dalla sua curiosità intellettuale e imprenditoriale.

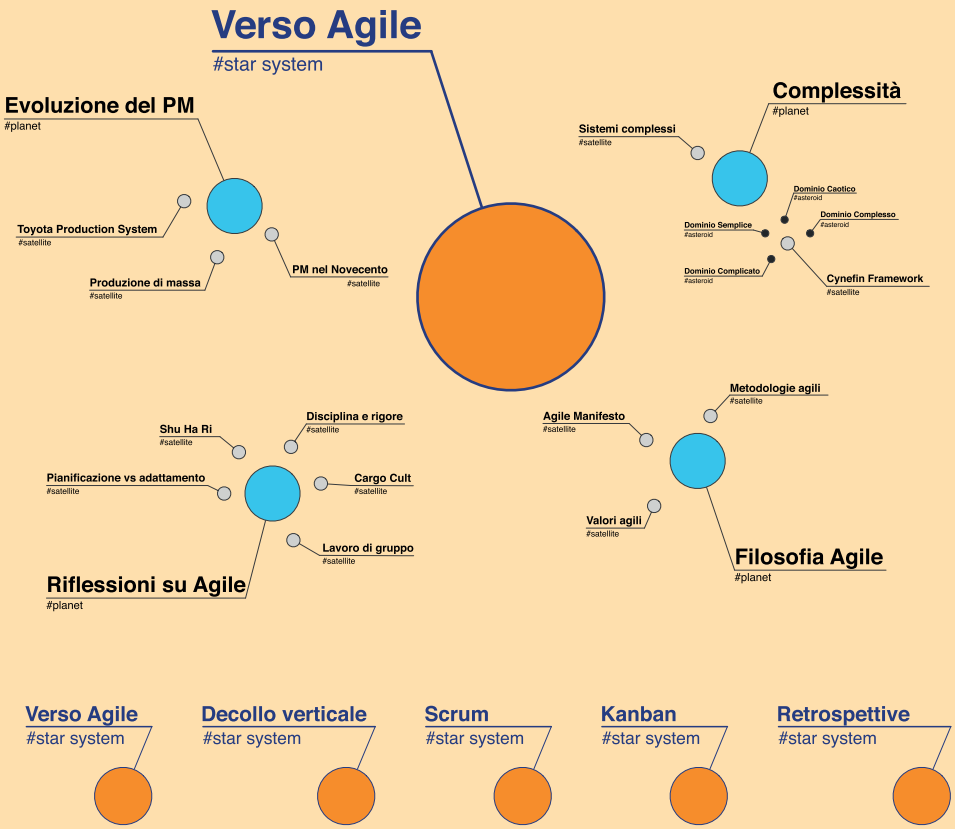
Ringrazio **Francesco Saliola** per aver pazientemente impaginato questo libro: come i tecnici del suono che ai concerti lavorano lontano dai riflettori, chi sta in redazione svolge un lavoro importante e tutt'altro che facile. Senza di lui questo libro non sarebbe così bello.

Oggi, dopo molto tempo in cui il libro è rimasto gratuitamente disponibile online, abbiamo deciso di produrre la versione cartacea della *Guida Galattica per Agilisti* ulteriormente riveduta e aggiornata. Spero che questo libro vi possa piacere: noi ci abbiamo messo, per quanto possibile, il nostro sapere, l'esperienza sul campo e, anche, molta passione.

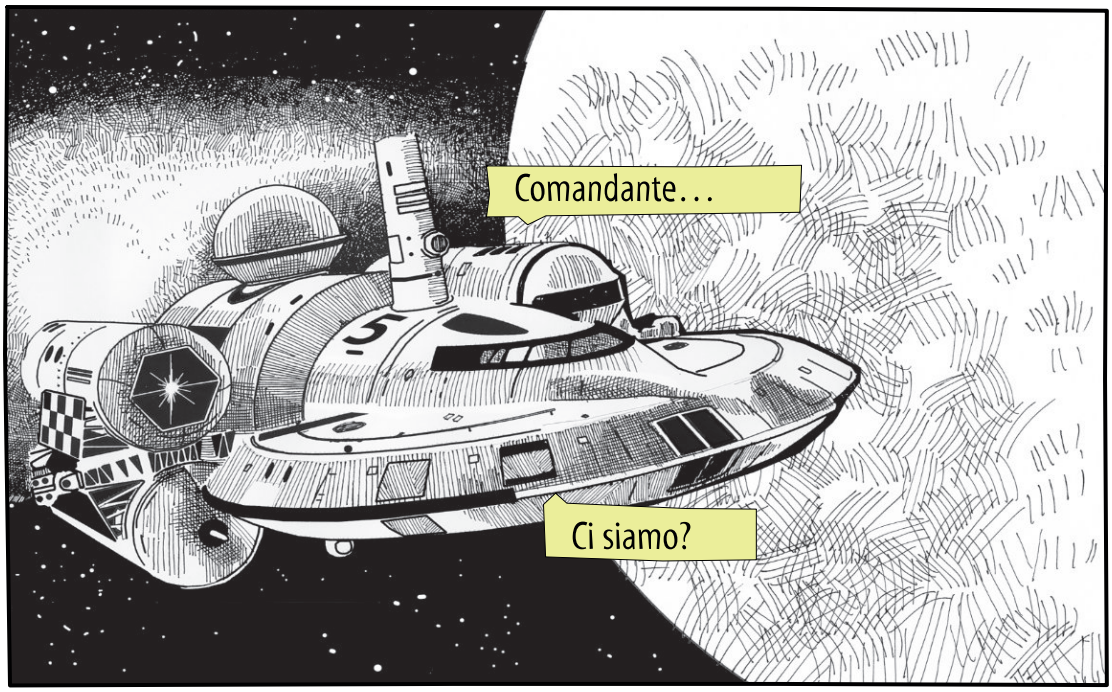
Giovanni Puliti

PARTE I
VERSO AGILE

Giovanni Puliti



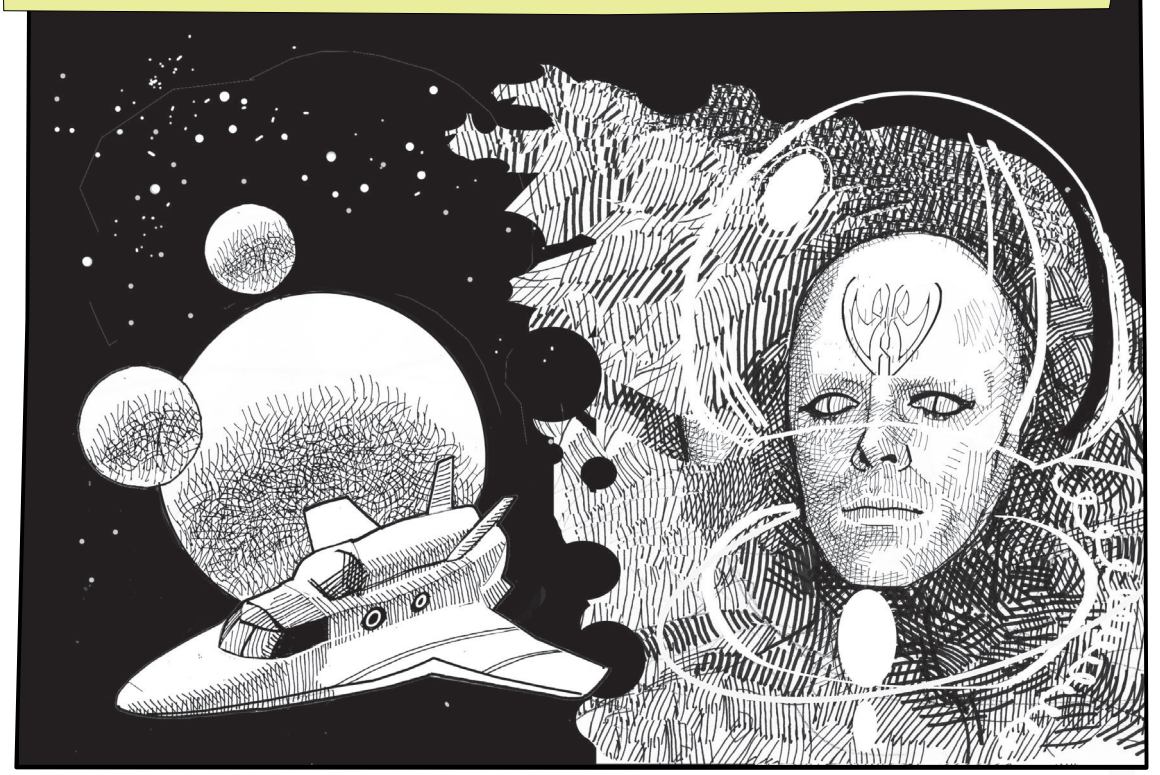
Parte 1
Verso Agile




Si, siamo appena usciti da velocità iperluce, stiamo per entrare nel sistema solare Agile.
Connetti il radiofaro, avvia la procedura di avvicinamento.

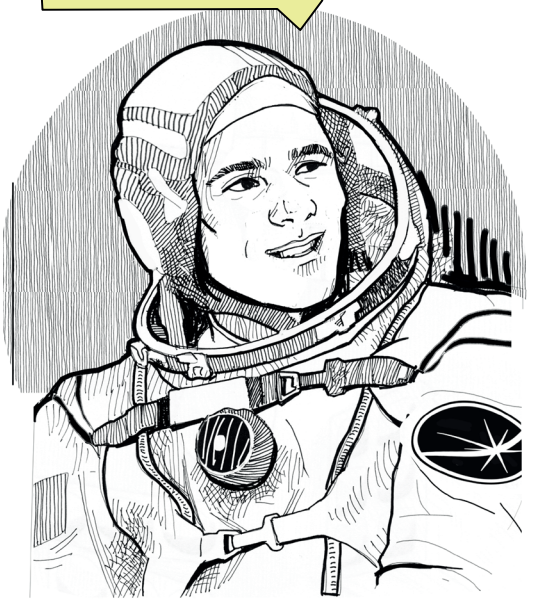

Non le nascondo che sono piuttosto impaziente di imparare i fondamenti del movimento Agile.

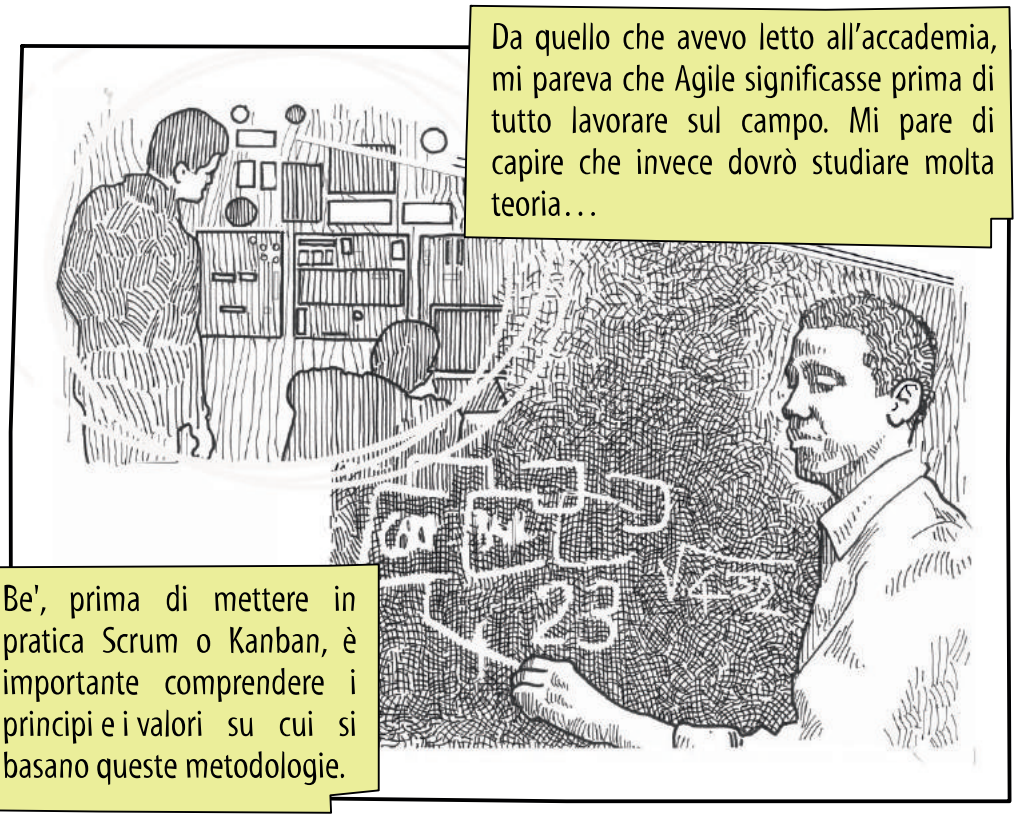
Non essere impaziente, tutto a tempo debito. Il cammino è appena iniziato. Per imparare a usare le metodologie agili dovremo visitare i vari pianeti del sistema Scrum e magari fare un giro con gli speedster sulle sue lune.

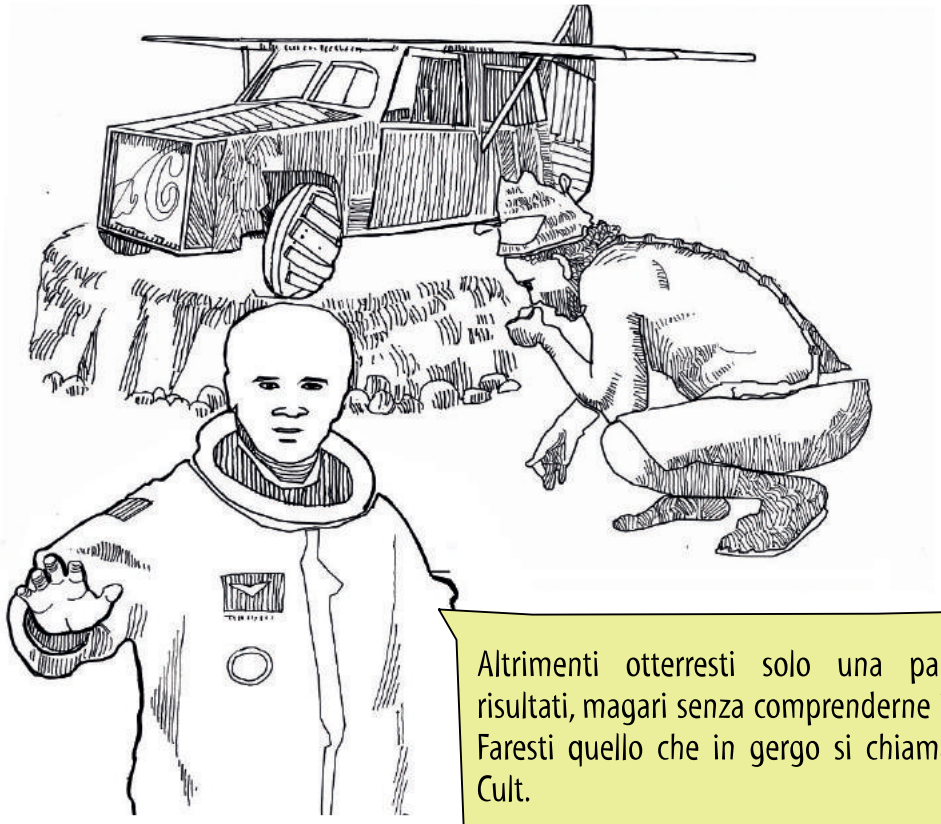


Allora cosa impareremo dalla visita a questo sistema?

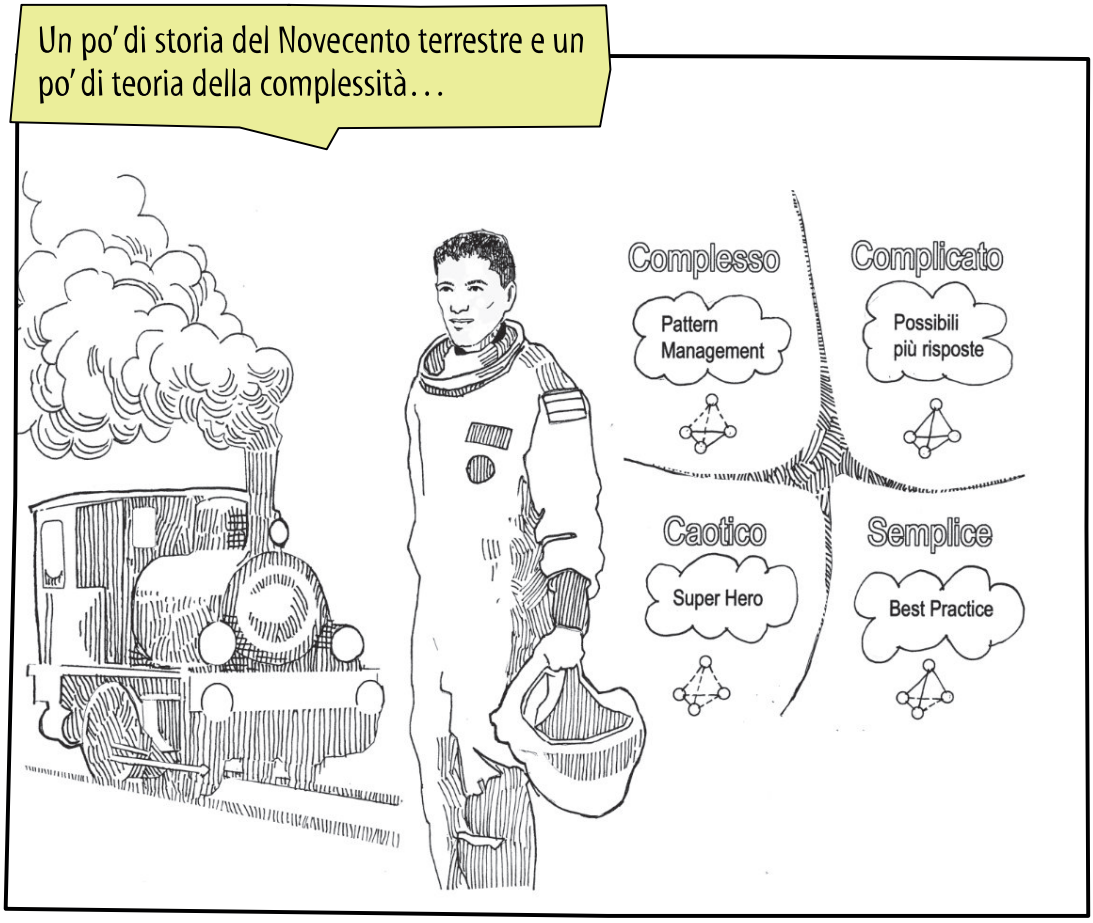
Poi entreremo nel sistema Kanban.



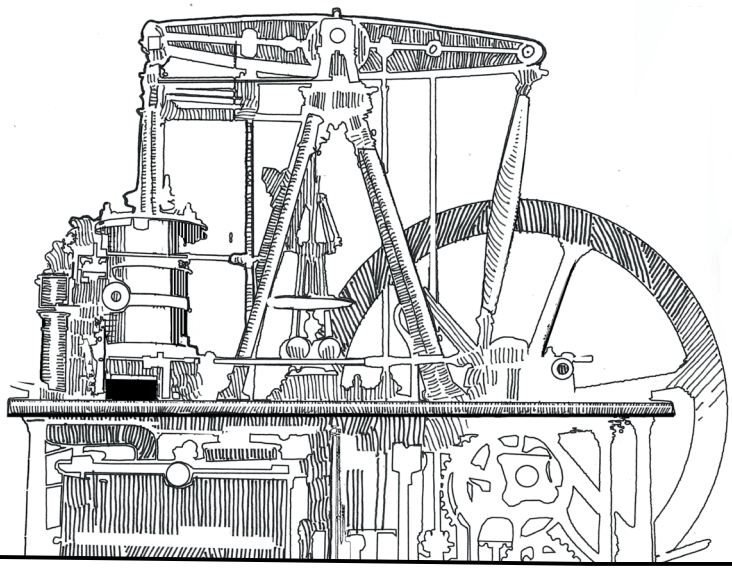

Da quello che avevo letto all'accademia, mi pareva che Agile significasse prima di tutto lavorare sul campo. Mi pare di capire che invece dovrò studiare molta teoria…
Be', prima di mettere in pratica Scrum o Kanban, è importante comprendere i principi e i valori su cui si basano queste metodologie.


Altrimenti otterresti solo una parte dei risultati, magari senza comprenderne i motivi. Faresti quello che in gergo si chiama Cargo Cult.


Quindi cosa mi devo aspettare dalla visita su questo pianeta?


Un po' di storia del Novecento terrestre e un po' di teoria della complessità…
Complesso
Pattern Management
Complicato
Possibili più risposte
Caotico
Super Hero
Semplice
Best Practice

La rivoluzione industriale e l'evoluzione tecnologica, che si verificò in alcuni paesi della Terra per tutta la durata del XX secolo, caratterizzarono fortemente la scuola di pensiero e la nascita delle discipline di gestione classiche come oggi le conosciamo.

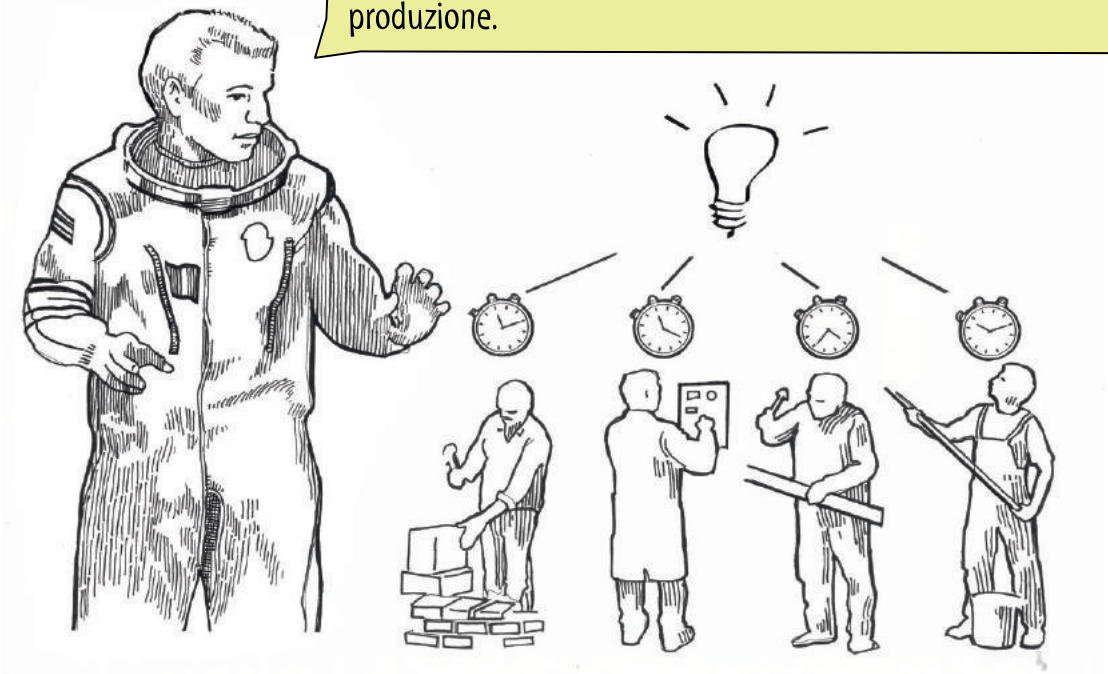


In poco più di cento anni si passò da una società rurale a una fortemente industrializzata, fino ad arrivare all'era dell'atomo, di internet e delle biotecnologie.

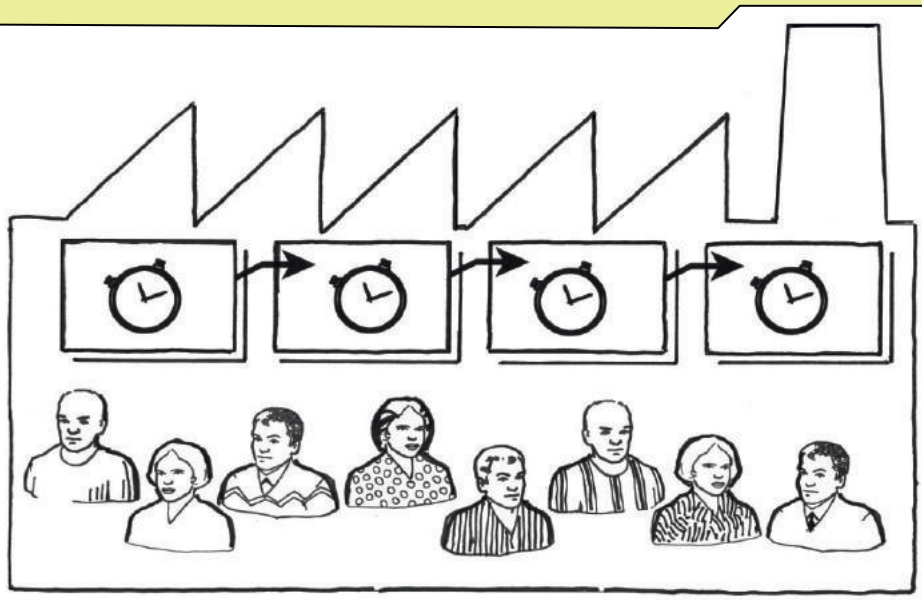
Fino alla Seconda Guerra Mondiale, il modello organizzativo utilizzato è quello oggi noto come micromanagement: poche persone possedevano le informazioni e la conoscenza sul cosa fare e su come farlo.
Quei pochi dettavano gli ordini alle masse operaie che erano tenute a eseguire senza fornire un contributo attivo sul lavoro da svolgere.

Con gli anni, una serie di fattori, fra cui il progresso tecnologico, le lotte sindacali, l'emancipazione sociale e l'aumento del benessere, modificarono le condizioni che avevano reso possibile quel tipo di impostazione, anche se ci volle ancora molto tempo per abbandonare quel tipo di organizzazione.

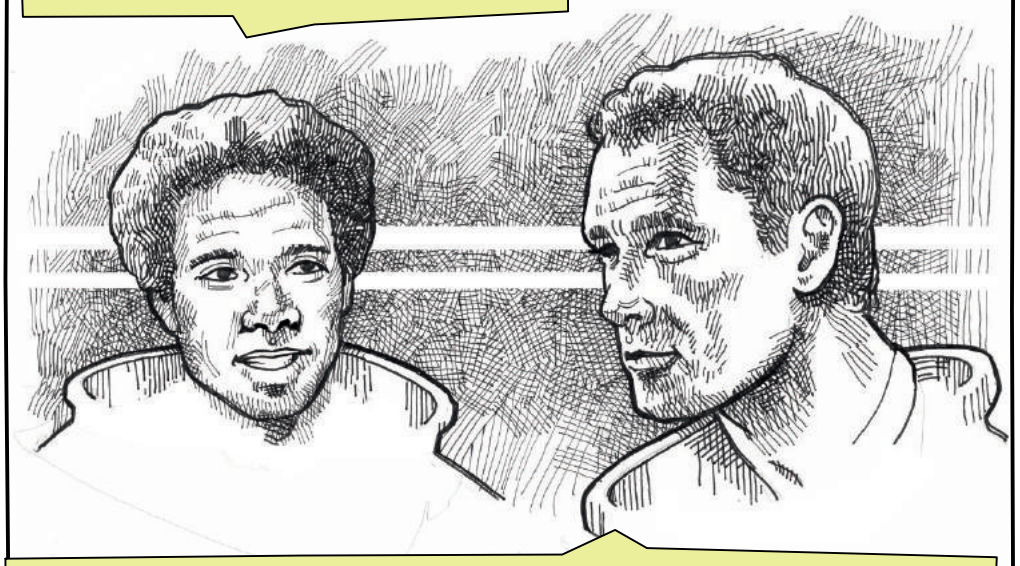
Si radicò fortemente l'idea che, per svolgere un lavoro, ad esempio costruire un'auto, si dovesse partire prima dallo stabilire tutte le fasi del lavoro, stimarle, pianificarle e poi controllarne la produzione.



In quel periodo, l'approccio deterministico era quello che forniva i migliori risultati: si scomponeva un processo in sotto-attività che venivano assegnate agli esecutori. Meno tempo si impiegava in ogni fase, migliore era la performance complessiva.

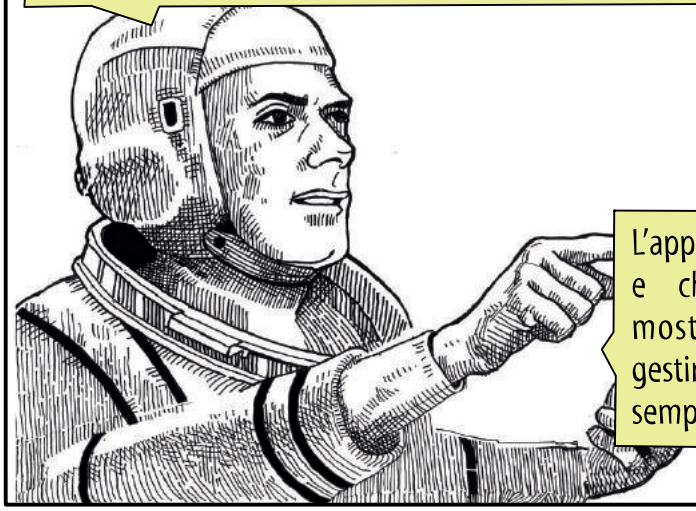


Ma questo modo di lavorare, da quel che mi ricordo dagli studi, si è poi mostrato piuttosto inefficiente, no?



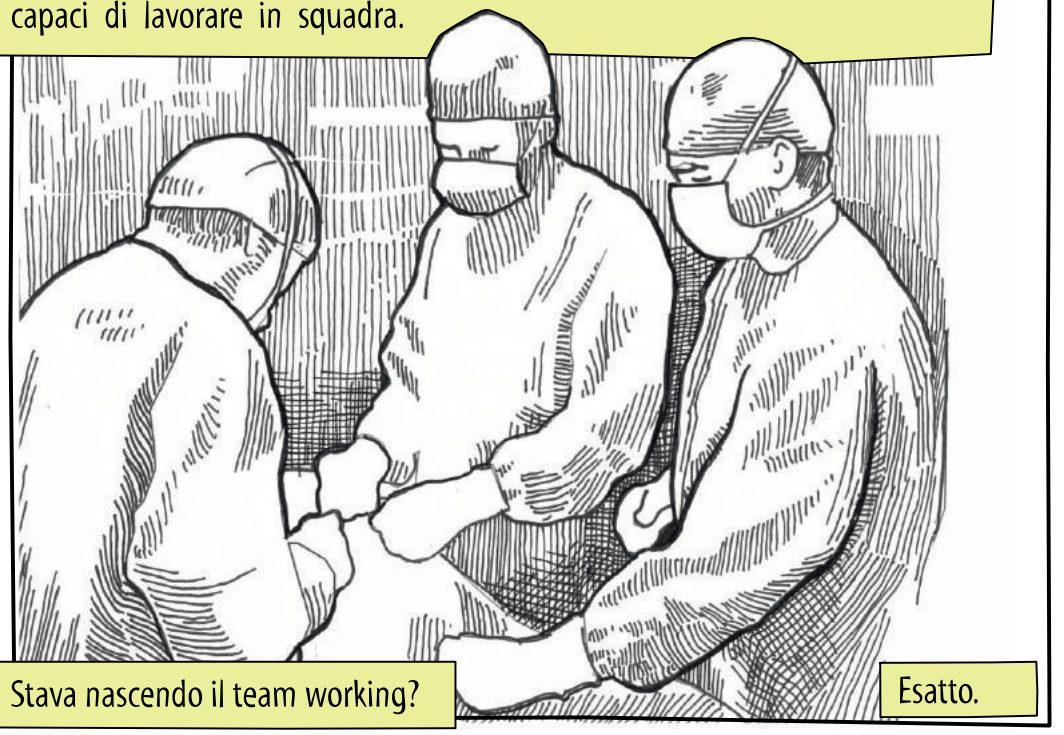
Sì, successivamente sì. Però non bisogna dimenticare il contesto storico in cui questo modo di lavorare si formò: per il tipo di attività lavorative della prima parte del secolo era adatto e anche efficiente.

Solo dopo la Seconda Guerra Mondiale le cose cominciarono a cambiare: la complessità gradualmente aumentò sempre più. In questo periodo, quell'approccio iniziò a mostrare tutti i suoi limiti.



L'approccio del capo che "sa" e che "ordina" iniziò a mostrarsi incapace di gestire una complessità sempre crescente.

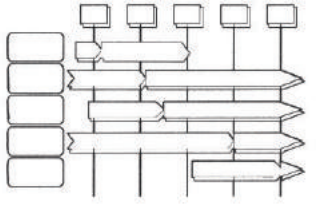
Si cominciò a pensare che fosse necessario cambiare il modo di lavorare, muovendosi verso un modello organizzativo basato su gruppi di persone che avessero competenze specifiche differenti e che fossero capaci di lavorare in squadra.

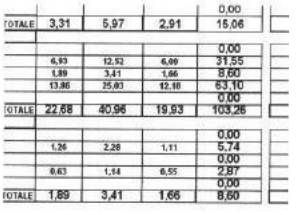


Stava nascendo il team working?

Esatto.

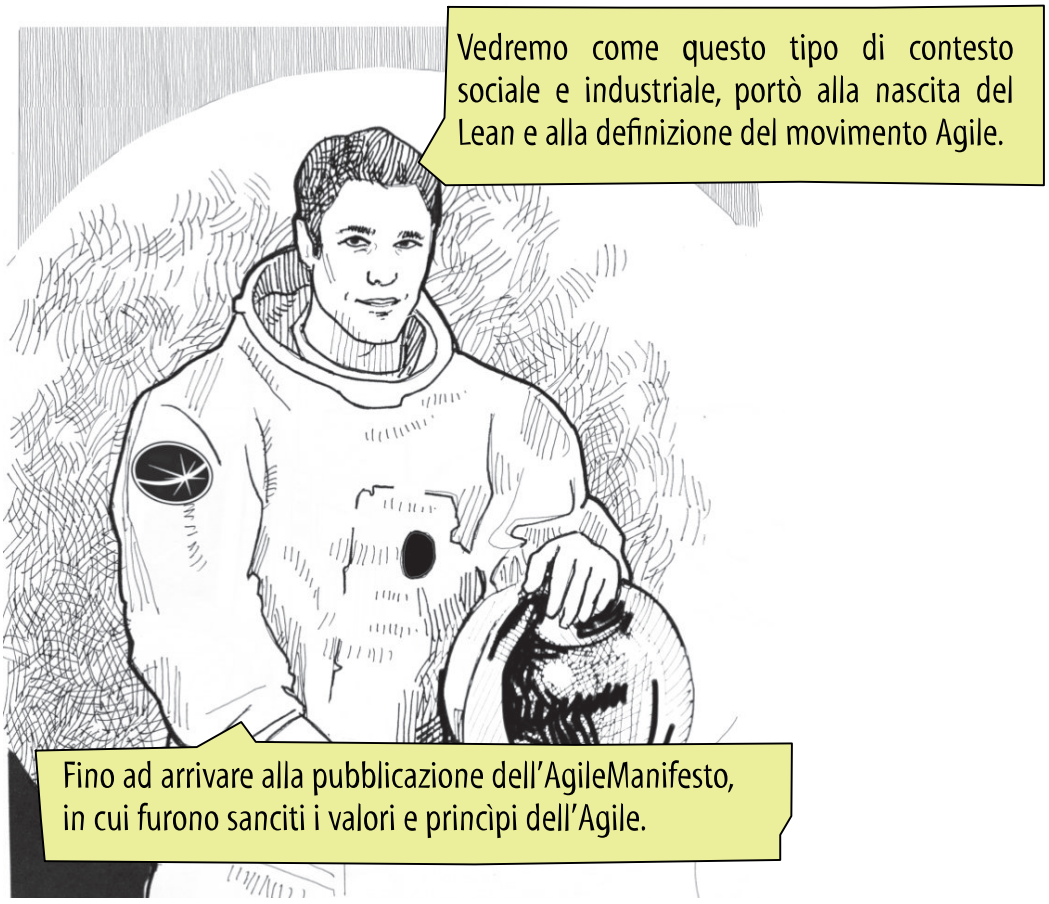
Inoltre si cominciò a mettere in discussione l'approccio deterministico. In molti si resero conto che non era più efficace voler pianificare in dettaglio tutte le varie fasi della lavorazione prima ancora di iniziare a lavorare.





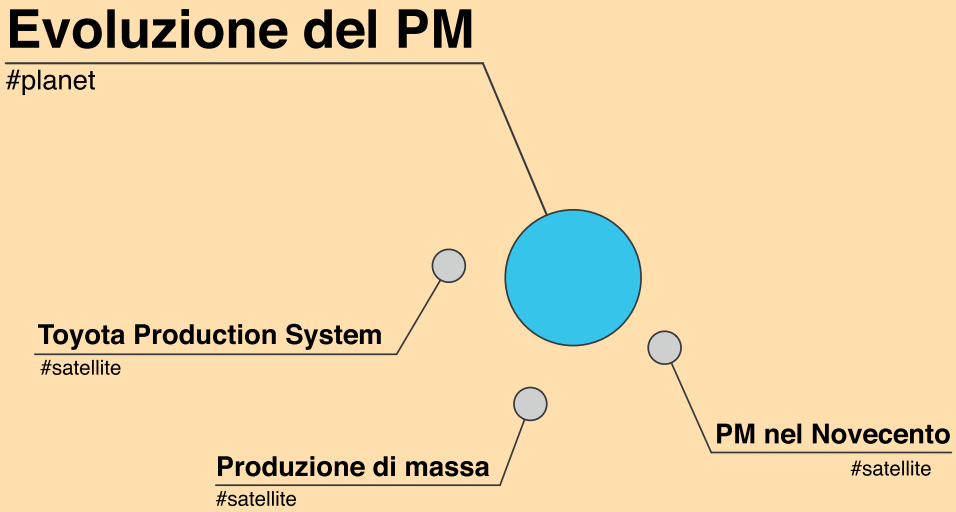
				0,00
TOTALE	3,31	5,97	2,91	15,06
				0,00
	6,93	12,52	6,00	31,55
	1,89	3,41	1,66	8,60
	13,86	25,03	12,18	63,10
				0,00
TOTALE	22,68	40,96	19,93	103,26
				0,00
	1,26	2,28	1,11	5,74
				0,00
	0,63	1,14	0,55	2,87
				0,00
TOTALE	1,89	3,41	1,66	8,60

Si vide che troppi erano i fattori influenzanti il comportamento di un sistema complesso: poco a poco si comprese quanto fosse inutile accanirsi in attività di pianficazione, previsione e controllo.


Vedremo come questo tipo di contesto sociale e industriale, portò alla nascita del Lean e alla definizione del movimento Agile.
Fino ad arrivare alla pubblicazione dell'AgileManifesto, in cui furono sanciti i valori e princìpi dell'Agile.


Fu quella la scintilla che diede vita a un nuovo modo di lavorare, fatto di pratiche, di metodologie e di processi. Scrum, XP e Kanban sono basati su quei valori e quei principi.

Capitolo 1. L’evoluzione del project management: dalla produzione di massa al Lean



Introduzione

La storia industriale del Novecento è caratterizzata dal susseguirsi di innovazioni tecnologiche, di cambiamenti organizzativi e di rivoluzioni gestionali, concretizzate nella crescita dei sistemi produttivi, nell'aumento della complessità delle organizzazioni, nel tentativo di migliorare sia le tecniche organizzative che i processi di produzione.

Tali cambiamenti si manifestarono, nella prima metà del secolo, in maniera più evidente nel **settore automobilistico**, specialmente negli Stati Uniti e in Giappone. L'opera di alcuni industriali, Henry Ford *in primis*, favorì negli USA il passaggio da un modello aziendale di tipo artigianale a quello poi diventato famoso come **processo** di **produzione di massa**. Analogamente e nello stesso periodo, in Giappone, alcune idee altrettanto innovative si svilupparono in seno alle aziende della famiglia Toyoda, dando vita al **Toyota Production System**, precursore della cosiddetta **produzione snella** (**Lean Production**).

In questo capitolo ripercorriamo quanto accaduto nel "secolo breve" attraverso il racconto dei fatti, dei protagonisti, delle idee, dei successi e dei fallimenti; cercheremo di evidenziare il percorso che ha dato vita a un certo modo di intendere la gestione di progetto, noto oggi come **micromanagement**, nato in seno al mondo dell'industria pesante ma che poi si è diffuso in altri settori come quello della produzione del software.

Le idee di **Ford**, basate sulla definizione di regole, controlli e processi deterministici, furono per l'epoca molto **innovative** e permisero inizialmente una fortissima **crescita**. A partire dal secondo dopoguerra, quelle stesse idee che si erano affermate come innovative ed efficaci iniziarono a dimostrarsi inadatte a un mondo che stava cambiando. Lentamente, ma inesorabilmente, l'era dei **sistemi chiusi**, della **scarsa concorrenza**, dei **mercati stabili**, cominciò a lasciare il posto a un'epoca fatta di nuovi attori in forte **concorrenza** fra loro, di **mercati globali** in costante e rapida evoluzione, di sistemi **complessi altamente interconnessi fra loro**.

Lo scopo di questo capitolo è quindi quello di raccontare come mai, verso la fine del secolo scorso, si sia sentito il bisogno di mettere in discussione il **project management classico** (quello del **micromanagement**, della **scomposizione** delle attività e dei processi **deterministici**); cercheremo di capire da cosa sia nata la **filosofia Lean** e come si sia sviluppato il **movimento Agile**.

A supporto delle analisi e delle valutazioni che faremo in questa parte, ampio spazio sarà dato al tema della **complessità**, argomento fondamentale per comprendere processi e organizzazioni: è infatti grazie allo studio dei problemi affrontati dalla **teoria della complessità**, che si possono capire meglio le dinamiche e le trasformazioni delle organizzazioni, delle aziende, del mondo produttivo.

L'evoluzione del management

L'evoluzione del management nello scorso secolo ha subito diversi impulsi, grazie al lavoro di noti studiosi formatisi nell'Ottocento, come **Taylor** (1856-1915) e **Fayol** (1841-1925), o pienamente inseriti nel "secolo breve", come **Drucker** (1909-2005) e

Deming (1900-1993), quest'ultimo noto soprattutto per la formulazione delle 14 regole sulla qualità totale che hanno portato al **Lean Management** e al **Toyota Production System** [TPS].

Si è trattato di un processo di continua ricerca, con prove ed errori, deduzione e induzione, test su cosa potesse funzionare meglio e scoperte di cosa invece fosse del tutto inappropriato. Questo percorso fu animato fin da subito dalla volontà di creare il modello universale di gestione (intesa come **organizzazione** e **controllo**) e di previsione, che si potesse applicare all'industria produttiva, alle aziende fornitrici di servizi, ma anche, in senso più ampio, a corpose fette di organizzazioni e gruppi sociali. L'obiettivo si dimostrò fin da subito molto ambizioso ma difficilmente realizzabile; nacquero in quel periodo diversi strumenti e svariate metodologie che però poco si prestavano a un'adozione globale, che desse risultati soddisfacenti in ambiti completamente differenti; questo accadeva essenzialmente a causa di due problemi: **specificità** e **mancanza di strategia**. Molti infatti, se adoperati come strumenti universali, soffrivano delle limitazioni derivanti dalla specificità del contesto dal quale erano stati estrapolati; contemporaneamente quasi tutti offrivano un approccio tattico (soluzione tampone per l'immediato) e quindi erano inadatti ad essere utilizzati come strumenti per la definizione di strategie di lungo termine.

Gli studi che seguirono, inoltre, hanno evidenziato che gli studiosi del secolo scorso, autori di molti dei modelli, sottovalutarono il **coefficiente di complessità** di un sistema o dell'ambiente esterno in cui tale sistema si trova a operare. Questa carenza è forse da imputarsi a una mancanza di sensibilità su questi temi o, più probabilmente, all'assenza di una casistica concreta che sistematizzasse sistemi più o meno complessi.

In passato il management doveva gestire sistemi **lineari**, sistemi in cui **causa** ed **effetto** sono legati in modo prevedibile, e dove il processo è ripetibile indefinitamente; oggi diremmo che, all'epoca, la maggior parte delle organizzazioni operava in contesti **semplici** o al tuttalpiù **complicati** (si veda, al capitolo successivo, il modello Cynefin sui diversi gradi di complessità).

Con il passare del tempo, fattori come la crescita dei volumi (persone, merci, servizi, interscambi), la nascita dei mercati globali, il progresso scientifico, industriale e sociale hanno portato a un mondo sempre più interconnesso, costituito da componenti in grado di svolgere compiti e di decidere **autonomamente,** ma che restano al tempo stesso **interdipendenti**. Quelle che un tempo erano organizzazioni solo complicate, si evolsero verso sistemi complessi dove il livello di complessità è maggiore che nei sistemi **complicati**.

Purtroppo la lingua italiana spesso confonde i due concetti che invece, come avremo modo di vedere nel capitolo dedicato a Cynefin, sono differenti fra loro e hanno un preciso significato nell'ambito di alcuni modelli interpretativi e organizzativi.

Breve storia del management

Gestire organismi di grandi dimensioni o coordinare attività molto strutturate non è una capacità nata con l'era moderna: basterebbe pensare, ad esempio, all'esercito

dell'antica Roma, capace di conquistare gran parte dell'Europa e del mondo mediterraneo. Tali conquiste furono possibili non tanto in virtù delle pur valide capacità di combattimento, ma grazie soprattutto a una perfetta organizzazione logistica, alla capacità di sfruttare al meglio le attività dei soldati anche nella costruzione di opere di ingegneria civile — come strade, ponti e acquedotti — nonché alla perfetta ripartizione di compiti e mansioni.

La scuola classica

In epoca moderna, il lavoro di **pionieri** della **scuola classica**, come Fayol o Taylor, ha fornito importanti spunti per gli studiosi che sono venuti dopo loro, avendo codificato gli elementi che hanno poi favorito la nascita del management moderno. Nonostante i vari percorsi di studio siano stati più o meno diversi, in un modo o nell'altro tutti giunsero alla formulazione di quelle che possono essere considerate le attività basilari e necessarie per pianificare e gestire un processo o un sistema (in sintesi il **management**).

Queste 5 attività sono oggi note come le cinque funzioni manageriali di Henry Fayol:

- *planning* (pianificazione);
- *organizing* (organizzazione);
- *staffing* (organizzazione e gestione del gruppo delle persone);
- *directing* (o *leading*, ossia guida del gruppo per perseguire gli obiettivi prefissati);
- *controlling* (verifica passo dopo passo che quello che si sta facendo corrisponda con gli obiettivi e le metriche prefissate).

La modalità con la quale le industrie e le aziende hanno messo in pratica queste discipline ha caratterizzato fortemente il management del Novecento: a seconda dello schema con cui le cinque funzioni di Fayol sono state organizzate (serializzandole o scomponendole in vario modo) si possono infatti ritrovare alcune pratiche di project management note anche ai manager contemporanei.

Organizzazioni e processo di produzione: dal modello verticale a quello orizzontale

Nel secolo scorso, la maggior pare delle aziende si sono organizzate in strutture verticali, orientate alla definizione prima, e al rispetto poi, di una rigida struttura gerarchica; in questo contesto spesso si è potuto assistere alla nascita di **divisioni** di competenza (modello che spesso sopravvive tutt'oggi) regolate con una netta **separazione** della funzioni e delle responsabilità.

In tal senso, molti degli sforzi erano volti ad accentrare ogni aspetto del processo di controllo e produzione: un esempio piuttosto famoso è quello di Ford, che si impegnò per inglobare nelle sue aziende l'intero processo produttivo. Le industrie di Ford arrivarono ad acquisire le cave per l'estrazione dei metalli e le fonderie per la loro lavorazione, le fabbriche per la produzione dei componenti, fino, ovviamente, agli stabilimenti per la fabbricazione delle automobili. Nell'ottica fordista della prima metà del Novecento, si riteneva vantaggioso poter controllare completamente ogni elemento del processo di

produzione: era opinione comune che questo permettesse una totale autonomia operativa, consentendo di aumentare la produzione e riducendo i costi.

Come avremo modo di vedere, tali convinzioni si basavano però su una serie di ipotesi fallaci, prima fra tutte quella derivante dalla mancanza di una reale competizione: Ford era allora il più forte, se non l'unico, produttore di massa del settore automobilistico, cosa che gli permetteva di imporre le sue regole sia nei confronti dei consumatori (ridotti a meri acquirenti senza possibilità di scelta) sia nei confronti della propria struttura di vendita (i concessionari erano distributori autonomi dalla casa madre, ma soggetti a leggi "capestro" sui volumi di auto da acquistare e rivendere). Grazie a quel modello **verticale**, egli poteva infatti massimizzare il controllo interno ed esterno; in assenza di una reale competizione, poi, era possibile non preoccuparsi dei difetti insiti nella sua natura: fortissimi elementi di **spreco**, quindi bassa efficienza, scarsa flessibilità, disinteresse agli eventuali cambiamenti del mercato che, in regime di sostanziale monopolio, erano inesistenti o comunque pilotati da Ford stesso.

Nel Secondo dopoguerra, invece, vari fattori posero le condizioni per la creazione di una reale concorrenza, fatta di alternative per il consumatore, di vari produttori in competizione fra loro, di nuovi mercati internazionali. A poco a poco iniziarono a venir meno quei presupposti su cui il modello verticale aveva costruito il proprio successo: cominciò così un'inversione di tendenza in cui le aziende iniziarono a organizzarsi in modo differente, ponendo sempre più l'attenzione su quello che oggi viene identificato come il **core business** dell'impresa.

Le grosse aziende iniziarono a esternalizzare parte della produzione dei componenti che poi venivano assemblati in fabbrica, a cedere la proprietà di quelle risorse e di quei servizi che potevano essere forniti da produttori esterni; proprio grazie a una forma di concorrenza in scala, infatti, questi potevano garantire maggior efficienza, traducibile in minor costi, tempi di consegna minori, qualità più alta.

Contemporaneamente si iniziò lavorare per identificare ed **eliminare** gli **sprechi** in tutte le forme in cui questi potevano manifestarsi; si cominciò quindi a lavorare a una massiccia revisione dei processi di produzione, a investire costantemente in nuovi macchinari, a indagare nuove opportunità di mercato, a spostare parte della produzione dove si ritenesse fosse più conveniente.

La Seconda Guerra Mondiale rappresenta probabilmente l'evento che in modo più forte ha dato impulso a questa trasformazione. Al termine del conflitto, infatti, accanto a Paesi completamente annientati, ne troviamo altri che uscivano vincitori e che quindi si trovavano con un settore industriale al massimo della potenza produttiva perché fino a poco prima aveva supportato la macchina bellica. Per gli Stati Uniti d'America, in particolare, questo si tradusse da un lato nella necessità di riconvertire processi e prodotti, dall'altro nel dover trovare nuovi mercati dove "piazzare" le proprie merci. Nuovi attori iniziarono ad affacciarsi sui mercati, realizzando le prime forme di concorrenza.

Stavano nascendo le condizioni per il mercato globale come oggi lo conosciamo.

Ford vs Toyota: storia di due dinastie industriali

Non è possibile comprendere la nascita del sistema di produzione conosciuto come **Lean** senza uno sguardo alla storia dell'industria automobilistica nel secolo scorso, dove spiccano come protagoniste le due più importanti case di produzione statunitensi, **Ford** prima e **General Motors** dopo, e la loro principale concorrente, la giapponese **Toyota**.

Questa storia mostra lo sviluppo del modello industriale creato da Henry Ford, in contrapposizione con quanto invece fatto da alcuni manager giapponesi, primo fra tutti **Taichi Ōno**, fedele collaboratore della famiglia Toyoda. Questa storia, inoltre, ci illustra i principi teorici e i valori alla base di due modi di intendere l'industria automobilistica (e non solo): da un lato la produzione di massa fordista, dall'altro la produzione leggera o **Lean Production**.

Tra le varie fonti cui fare riferimento, la più importante è probabilmente il libro *La macchina che ha cambiato il mondo* [MCM], un testo che è a metà fra un romanzo e un saggio scientifico. Per ovvi motivi di spazio, è stata inserita in questo capitolo solo una sintesi di tutto quanto è avvenuto nell'arco di poco più di cento anni (dal 1903 al 2008) e che è raccontato in modo molto più accurato e completo nel testo succitato. Per chi fosse invece interessato a qualcosa di fruibile in modo più rapido, consigliamo il videodocumentario sulla storia della famiglia Toyoda e della casa automobilistica da loro fondata [TGS].

La storia della Toyota, prima parte: nascita del settore automobilistico

Verso la fine dell'Ottocento la famiglia Toyoda, con a capo Sakichi Toyoda, rappresentava una delle industrie più potenti e ricche del Giappone, grazie a una florida produzione di stoffe. Le loro fabbriche disponevano di innovativi ed esclusivi sistemi di produzione; buona parte delle macchine era costruita sulla base di brevetti che la famiglia stessa deteneva.

Verso i primi anni del Novecento, a causa di una congiuntura di fattori esterni e di una crisi economica che attanagliava buona parte del Paese, Sakichi Toyoda si trovò a dover chiudere molti stabilimenti. Sull'orlo del totale fallimento, la famiglia Toyoda si trovò a dover ricominciare dal nulla.

In quegli anni difficili per l'azienda, Sakichi fu affiancato dal nipote, Kiichiro Toyoda, che avrebbe guidato l'azienda fino alla Seconda Guerra Mondiale. Kiichiro si distinse fin da subito per un'innata propensione all'innovazione e alla continua ricerca di un **miglioramento** da portare all'interno dell'azienda. Oltre a progettare un piano di rilancio per le proprie industrie tessili, in quegli anni Kiichiro iniziò a chiedersi se fosse il caso di investire in **settori** completamenti **nuovi**, in modo da cogliere le nuove opportunità offerte dal secolo che stava iniziando.

Fu in quel periodo che, animato da uno spirito di cambiamento misto a una profonda curiosità, Kiichiro intraprese un viaggio negli USA, alla ricerca di nuove idee e modelli industriali a cui ispirarsi. Quel periodo era particolarmente importante per l'industria degli Stati Uniti d'America, in particolare per il settore automobilistico: stavano

iniziando a nascere i primi esempi di quel modello industriale che poi abbiamo imparato a chiamare **produzione di massa**.

Come avremo modo di vedere, quel viaggio fu estremamente importante per Kiichiro, dato che radicò in lui la convinzione dell'importanza di investire nel **settore automobilistico**: di lì a poco sarebbe partito il progetto industriale automobilistico che avrebbe portato in pochi anni alla creazione del primo modello di automobile con marchio **Toyota**: il cambiamento del nome, con una "t" al posto di una "d", fu il frutto di una operazione di marketing.

Il settore automobilistico in USA a inizi Novecento: dal modello artigianale alla produzione di massa

Quando nel 1908 Kiichiro Toyoda fece il suo primo viaggio negli USA, nel settore automobilistico si stavano consolidando le prime esperienze industriali, che pian piano prendevano il posto delle piccole aziende a impostazione artigianale. In queste **officine artigianali**, spesso a conduzione familiare, si trovava una forza-lavoro altamente specializzata: forgiatori, saldatori, battilastra, molatori, falegnami e tappezzieri seguivano l'**intero processo** di produzione, dalla progettazione all'assemblaggio. Spesso queste piccole fabbriche nascevano come trasformazione o evoluzione di aziende derivate da altri settori, principalmente da quello ciclistico, ma anche dall'idraulica industriale e dalla meccanica pesante.

Il processo prevedeva la **lavorazione manuale** delle varie parti dell'automobile con una limitata automazione con l'uso di macchine: i sistemi di pressatura e forgiatura erano rudimentali e offrivano una bassissima precisione nella lavorazione, tanto che spesso i pezzi dovevano essere poi rifiniti o rilavorati a mano; fra un prodotto finito e un altro spesso vi era un'evidente differenza sia per l'aspetto che per le dimensioni.

La clientela tipica di queste aziende era composta dalla ricca borghesia che poteva permettersi di pagare e di apprezzare un prodotto curato dei minimi dettagli, personalizzato negli accessori e negli allestimenti. L'attesa necessaria per avere un'auto di questo tipo non era un problema, dato che all'epoca l'automobile era ancora un bene di lusso non necessario per la vita di tutti i giorni.

In questo contesto iniziò a operare **Henry Ford** il quale, nell'arco di tutta la sua carriera professionale, cercò di trasformare questo approccio artigianale lento e costoso in qualcosa di completamente differente, più efficiente e rapido: la **produzione** di **grandi quantitativi** di prodotti finiti.

La sua idea era di produrre un'automobile a basso costo, che fosse accessibile alla massa delle persone e non solo ai ricchi. La qualità del prodotto finito non fu quasi mai un problema di cui Ford volle preoccuparsi. Tutti i suoi sforzi nel corso degli anni furono indirizzati a perseguire questi obiettivi: la totale **standardizzazione del processo** e l'**uniformità del prodotto**.

Questo processo di cambiamento cominciò a mostrare i primi risultati evidenti quando, nel 1908, Ford dette vita alla produzione del famosissimo **modello T**, che sarebbe

stato realizzato per quasi venti anni in circa 15 milioni di esemplari. L'idea vincente fu quella di creare un'auto facile da costruire, da guidare e da riparare; chiunque, con semplici strumenti, forniti in kit insieme all'auto, avrebbe potuto risolvere la maggior parte degli inconvenienti o dei guasti: dalla rimozione delle incrostazioni sulle teste dei cilindri allo spurgo dell'acqua dal serbatoio della benzina.

Oltre alla sua semplicità, il successo dell'auto era dovuto anche alle innovazioni apportate da Ford in fabbrica con l'intento di ridurre i costi e aumentare la produzione.

Il primo passo fu di realizzare, in fase di costruzione, l'**intercambiabilità** completa delle parti dell'auto in modo da semplificare il processo di assemblaggio e l'omogeneizzazione del prodotto finito: in questo modo infatti si potevano ridurre sia i tempi di assemblaggio che risparmiare sui costi. La semplificazione del processo di montaggio poté permettere l'eliminazione di **montatori qualificati** che avevano sempre rappresentato il grosso della forza-lavoro di tutte le aziende automobilistiche. Stranamente, queste modifiche nel processo di produzione passarono quasi inosservate agli occhi degli industriali e degli studiosi dell'epoca, i quali non ne compresero i benefici e le ricadute economiche.

L'intercambiabilità fu ottenuta grazie all'unificazione dei processi di calibratura, ma soprattutto grazie all'introduzione di nuovi macchinari frutto dei progressi tecnologici dell'epoca: in fabbrica si potevano finalmente utilizzare **macchine utensili** capaci ora di lavorare **metalli pretemprati**.

Altra innovazione introdotta da Ford fu quella di ridurre il **ciclo** di lavorazione di ogni operario, ossia il numero di operazioni che dovevano essere svolte su un'auto, prima di passare all'auto successiva. Precedentemente, in fabbrica il tempo medio di lavorazione era di circa 514 minuti, ossia 8 ore e 56 primi. L'intercambiabilità dei pezzi fu il primo passo che permise di ridurre tale periodo: eravamo agli albori del lavoro a **catena di montaggio spinto**.

Semplificare le mansioni di ogni singolo operaio, oltre a migliorare la produttività, permetteva anche di utilizzare una forza-lavoro composta da operai con livelli minimi di alfabetizzazione e con una preparazione tecnica praticamente inesistente. Questo si sposava perfettamente con la situazione socio-economica in essere in quel momento negli USA: nel 1915, negli stabilimenti Ford della **Highland Park**, gli operai provenivano da moltissimi Paesi esteri e parlavano più di **cinquanta idiomi**; molti, addirittura, non conoscevano nemmeno l'inglese. Ridurre la complessità di lavorazione, incrementare l'automazione e standardizzare le procedure di utilizzo dei macchinari era il modo per impiegare questa forza-lavoro in modo efficace.

La qualità nella produzione di massa: il modello push

Il processo di trasformazione messo in atto da Ford era animato, in modo quasi maniacale, da due obiettivi: **abbattimento dei costi** di produzione e **aumento dei volumi di produzione**. La mancanza di una reale concorrenza sul mercato e la possibilità quindi di imporre un proprio prodotto, quale esso fosse, furono le condizioni che negli anni permisero a Ford di concentrare i propri sforzi sulla "potenza di fuoco" (numero di auto

prodotte) piuttosto che sulla qualità intrinseca del prodotto finito. Nella produzione di massa, la **qualità** era infatti espressione soprattutto del **numero** di **pezzi prodotti**, non tanto della qualità del prodotto finito.

Questa visione sviluppatasi all'interno delle fabbriche Ford, investiva ogni fase della attività lavorativa; in seguito, gli studiosi di organizzazione aziendale e dei processi industriali hanno definito tale approccio come **modello push**. Che si trattasse di stringere un bullone, di assemblare un motore o di vendere un'auto presso il concessionario, ogni lavoratore doveva svolgere il proprio lavoro nel modo più rapido possibile, così da **spostare** il **pezzo** su cui si stava lavorando **verso** la **postazione successiva**; alle sue spalle un collega **premeva** (*push*) per passargli un nuovo pezzo da lavorare.

In questo scenario, ogni operaio doveva svolgere il più velocemente possibile il proprio lavoro, senza preoccuparsi di **valutare eventuali problemi** o **malfunzionamenti**. Se per esempio, durante l'assemblaggio di un componente, si fosse riscontrato un difetto evidente, si doveva procedere a sostituire il pezzo difettoso prelevandone un altro dal magazzino. Il pezzo difettoso veniva gettato in un cassone o direttamente per terra, senza che fosse analizzato per comprendere le cause del difetto; non era ritenuto interessante valutare se il malfunzionamento fosse da imputarsi a una cattiva gestione del processo di assemblaggio in fabbrica, oppure a un errore all'origine, nella costruzione del componente. Nella catena di montaggio, l'auto che prendeva forma non era mai testata prima del completamento della costruzione: per questo motivo molti problemi non evidenti non venivano identificati.

Eseguire i controlli durante l'assemblaggio avrebbe rallentato la produzione, cosa che era considerata un costo ingiustificato, anche a fronte della produzione di numerosi pezzi difettosi. Questa scelta introdusse la necessità di creare nuove procedure per la gestione delle auto difettose; queste, indipendentemente dal tipo di problema riscontrato, venivano reintrodotte in fabbrica per il controllo e l'eventuale riparazione.

Quello che oggi potrebbe apparire come un modello inefficiente, in realtà era compatibile con lo scenario dell'epoca. Per prima cosa è necessario valutare il mercato entro il quale Ford operava. Giacché monopolista del mercato, egli poteva decidere, e quindi imporre, quale fosse la soglia di **qualità accettabile** dal cliente finale, che non aveva altra possibilità di scelta. Dal punto di vista industriale Ford disponeva poi di una gran quantità di manodopera a basso costo, per cui poteva impiegare intere squadre di operai nel reparto di riparazione. La scarsa concorrenza e dinamicità del mercato inoltre gli permettevano di accatastare auto non funzionanti senza il rischio che la concorrenza potesse renderle obsolete con il rilascio sul mercato di nuovi modelli.

La nascita del management scientifico: il Taylorismo

Al fine di aumentare la produzione, ossia di ridurre i tempi di lavorazione per ciascuna auto, Ford notò che due erano i **fattori abilitanti**: il primo era legato ai **movimenti degli operati** utili al montaggio rispetto a quelli inutili; il secondo era il **frazionamento delle operazioni** da svolgere.

Nel primo caso si comprese l'importanza di evitare agli operai tutti quegli spostamenti che non fossero strettamente legati alla lavorazione "sul pezzo". Rientravano in questa categoria lo spostamento per reperire un pezzo da montare su uno scaffale, per andare a preparare un utensile o per spostarsi verso un nuovo componente da lavorare.

Il secondo fattore abilitante invece era legato alla **complessità della lavorazione** da eseguire. Solo dopo molte rivisitazioni del processo di produzione, si ottenne un risultato ritenuto ottimale: ogni operaio svolgeva **un solo compito elementare** come avvitare un bullone o inserire una vite in un buco della testa del motore. Dovendo fare una sola cosa tutto il giorno ogni operaio, anche senza alcuna formazione iniziale e con scarsa preparazione professionale, poteva fin da subito raggiungere elevati livelli di performance.

L'introduzione del **nastro trasportatore** a ciclo continuo consentì di realizzare questi obiettivi: si poteva infatti aumentare l'automazione e massimizzare il frazionamento, il tutto rendendo l'operaio immobile alla sua postazione di lavoro. Il **nastro** inoltre era un ottimo **sistema push**, il cui ritmo poteva essere cadenzato a piacere. Al termine del 1913 nei registri della fabbrica Highland Park [FHP] è riportato che il ciclo di lavorazione medio era passato da 514 primi a 2,3 minuti, poi a soli 1,19 minuti.

In tale contesto la fabbrica veniva gestita tramite un **processo di controllo deterministico**: i manager usavano statistiche e tempi di lavorazione di ogni singola operazione per prevedere o gestire la produzione dell'intero impianto. Stava nascendo quello che fu poi successivamente chiamato **management scientifico** o **Taylorismo** [SM], fondato sull'applicazione di modelli analitici basati su misure e decisioni economiche razionali. Il Taylorismo fu creato per gestire linee di produzione in cui lavorava una forza-lavoro non specializzata in un'America che stava crescendo economicamente, nelle cui fabbriche si utilizzavano squadre di operai immigrati privi di alcuna competenza specifica.

Ufficialmente l'obiettivo del Taylorismo era di fornire un modello a supporto dei lavoratori assicurandosi che venissero impiegati correttamente e che i datori di lavoro non abusassero delle loro prestazioni. La realtà è stata, purtroppo, radicalmente diversa.

Ai miglioramenti ottenuti grazie alla frammentazione delle operazioni, alla specializzazione dei cicli di lavorazione e all'introduzione della catena di montaggio spinta, si accompagnarono anche **nuovi costi** derivanti dalla necessità di introdurre nuove attività di controllo della produzione.

Ford si rese conto quasi subito che gli operai, ridotti ormai a qualcosa di molto simile a macchine operatrici, non avrebbero volontariamente collaborato per contribuire al miglioramento del lavoro o della qualità dei prodotti finiti; per questo non si aspettava che gli fossero comunicate informazioni sui problemi inerenti al processo di produzione o a difetti dei macchinari, né tantomeno si supponeva che gli operai avrebbero fornito suggerimenti o indicazioni su come migliorare il lavoro in fabbrica.

L'operaio era ormai un **esecutore di operazioni elementari** e quindi non aveva una **visione** del lavoro nel suo complesso, né tantomeno avrebbe avuto le capacità tecniche per gestirne l'avanzamento. Sorgeva quindi l'esigenza di introdurre nuove figure come il

tecnico del lavoro, il cui scopo era di coordinare il lavoro di tali operai, o del **controllore** o **capo reparto**, che invece doveva controllare che tutti svolgessero le proprie attività a dovere e che nessuno rallentasse il ritmo di lavoro.

L'estrema specializzazione degli operai inoltre portò alla nascita di un esercito di persone incaricate di svolgere tutto il **lavoro indiretto** come il riparatore, l'addetto alle pulizie e il ritoccatore, figure inesistenti ai tempi della produzione artigianale.

Ford portò lo stesso modello di specializzazione e divisione dei compiti anche negli uffici di progettazione, dando vita a una folta schiera di **tecnici** o **lavoratori di concetto**, i quali con gli anni avrebbero sempre più aumentato la distanza con la produzione: il loro lavoro infatti era quello di pensare, pianificare, progettare automobili che raramente avrebbero visto direttamente durante il procedimento di costruzione o assemblaggio.

La storia della Toyota, seconda parte: la nascita della produzione snella

Dopo l'avventura statunitense, Kiichiro rientrò in patria con molte ispirazioni ma poche idee sul come metterle in pratica. Conscio che quello automobilistico fosse il settore dove investire nel secolo che iniziava, era altresì convinto che quanto visto in USA non fosse replicabile *tout court* in Giappone, sia per le differenze in ambito economico, sia per quelle nel settore infrastrutturale che perfino per quelle sociali: tanto per fare un esempio, a differenza dagli USA, il Giappone non disponeva certo dei tanti lavoratori immigrati da impiegare come manovalanza di basso livello.

Ciò nonostante, fece partire questo suo nuovo progetto industriale, attivandosi in prima persona per reperire il denaro necessario ad aprire i primi stabilimenti industriali. La raccolta fondi fu resa possibile anche grazie alla vendita di numerosi brevetti sui macchinari tessili che avevano costituito buona parte della ricchezza di famiglia fino alla crisi dei primi anni del Novecento. Nei primi anni Trenta del secolo scorso nacque così quello che sarebbe diventato uno dei più importanti colossi del settore automobilistico prima del Giappone e poi in tutto il mondo.

Il percorso che portò la famiglia Toyoda (rinominata successivamente in Toyota) a essere la più importante dinastia industriale del Paese non fu semplice e privo di imprevisti: ciò nonostante, nell'arco di pochi anni e partendo da zero, i progettisti della Toyoda riuscirono a produrre il prototipo della **Toyota Model A1**, prototipo che di lì a poco (nel 1936) avrebbe dato vita alla **Toyota AA**, il primo modello destinato alla vendita. Considerando il punto di partenza della Toyota, la ridotta esperienza dei progettisti e le difficili condizioni economiche del Paese, ma soprattutto il pochissimo tempo impiegato, il progetto del modello AA è da considerarsi una sorta di miracolo imprenditoriale e ingegneristico.

Nel corso degli anni successivi Toyota iniziò la produzione in grande stile delle proprie auto, senza tuttavia eguagliare nemmeno lontanamente la potenza produttiva degli stabilimenti di Ford: quello che in Toyota si produceva in un anno, veniva prodotto in qualche giorno nello stabilimento di Rouge [ROU], il nuovo impianto inaugurato da Ford nel 1928 e che nel 1933 dava da lavorare a circa centomila persone.

La crescita della casa proseguì negli anni successivi in modo piuttosto costante fino alla Seconda Guerra Mondiale, evento che vide il Giappone uscire duramente sconfitto dagli alleati. I vincitori imposero regole e costumi tipici della società americana: fu introdotto, fra le altre cose, un sistema di **sindacati** e furono aboliti gli **Zaibatsu** [ZAI], le organizzazioni economiche di stampo feudale facenti capo alle famiglie storiche della società giapponese. Al loro posto le aziende furono obbligate ad adottare sistemi di azionariato e fu introdotto un nuovo modello economico basato sul **Keiretsu** [KEI]: ogni Keiretsu si reggeva sul finanziamento economico fornito da una o più banche. Sebbene entrambe le mosse mirassero a dare più potere al management, l'intento era quello di consentire in futuro alle aziende statunitensi di impadronirsi dei pacchetti azionari delle varie aziende. Come avremo modo di vedere, la naturale diffidenza per lo straniero da parte dei giapponesi sviluppò una serie di meccanismi di difesa che resero immune l'economia giapponese dal colonialismo economico degli Stati Uniti.

Ma l'introduzione dei sindacati e dell'azionariato avrebbe avuto importanti conseguenze per i personaggi di questa storia, e non tutto fu a vantaggio degli Stati Uniti. I sindacati giocarono un ruolo fondamentale quando, nel 1946, le conseguenze della guerra appena terminata comportarono una grossa crisi economica per il Giappone. In Toyota la crisi si manifestò in modo particolarmente violento: dopo lunghe e difficili trattative, Kiichiro ottenne di riorganizzare l'azienda in modo da scongiurare il fallimento. In base all'accordo con i sindacati, alla dirigenza fu concesso di licenziare più di duemila dipendenti, in cambio delle dimissioni di Kiichiro. Sempre in base all'accordo, nuove regole sarebbero state introdotte: nessun dipendente sarebbe stato più licenziato ma anzi avrebbe potuto lavorare nell'azienda per il resto della propria carriera professionale, regola che si sarebbe presto allargata a tutte le aziende del Giappone, dando vita alle cosiddette **aziende-famiglia**. Il livello di retribuzione sarebbe stato calcolato in base all'anzianità di servizio.

Con le dimissioni di Kiichiro, il timone dell'azienda passò nelle mani di suo cugino, Eiji Toyoda, figlio del fondatore Sakichi Toyoda; egli si trovò a dover gestire un'azienda in difficoltà composta da personale che non si poteva licenziare ma che avrebbe comunque percepito uno stipendio in costante crescita fino alla pensione. Eiji si avvalse della collaborazione di un ingegnere giapponese specializzato in meccanica, **Taiichi Ōno** che, con il tempo, sarebbe stato considerato il padre del sistema di produzione attuato nell'azienda automobilistica Toyota: il **Toyota Production System** o TPS [TPS]. Anche grazie alle idee di un altro importante personaggio che in quel periodo girava il Giappone facendo convegni e conferenze, **William Edwards Deming** (padre fra le altre cose del noto ciclo di miglioramento continuo, PDCA), Ōno iniziò a introdurre in azienda una serie di cambiamenti volti a stravolgere il concetto di **prodotto** e di **processo** di produzione.

In Toyota si impiegarono circa venti anni per mettere a punto e rendere efficiente il **TPS**: durante questo lungo periodo, l'obiettivo di Ōno fu sempre quello di operare per **migliorare** la **qualità** del prodotto finito e la **riduzione** degli **sprechi** durante la produzione. Ōno inoltre aveva a cuore un altro importante obiettivo: il rapporto con i dipendenti. In base ai nuovi accordi infatti l'azienda poteva contare su una manodopera fedele

ma la cui retribuzione sarebbe stata sempre più costosa. Ōno comprese che era necessario convertire un potenziale handicap in un importante punto a favore dell'azienda, dando vita a un programma di investimenti sulla **cultura professionale** dei dipendenti cercando per quanto possibile di coinvolgerli nei processi di controllo e di verifica.

Raccontare nei dettagli la storia di quel ventennio con le novità introdotte da Ōno, per quanto interessante, esula dagli scopi di questo capitolo: per questo, elencheremo solo alcuni dei punti più salienti.

Organizzazione in team cross funzionali

Una delle regole fondamentali alla base della rivoluzione di Ōno fu la costituzione di **team** in grado di svolgere **tutte** le **attività** di lavorazione all'**interno** della fabbrica. Ogni gruppo che lavorava alla costruzione delle auto era formato da persone appartenenti a due profili differenti, corrispondenti ai due contratti in vigore all'interno dell'azienda: **tecnico** e **assemblatore**. Il primo era di fatto un **ingegnere** che si preoccupava delle questioni che richiedevano competenze specifiche, mentre i secondi erano **operai** addetti alle operazioni più manuali. Dipendenti di entrambi i ruoli formavano gruppi autonomi costituiti al massimo da 10 persone; ogni membro del gruppo poteva svolgere una serie molto ampia di mansioni.

Ogni squadra, oltre a dedicarsi della costruzione delle autovetture, doveva inoltre occuparsi anche degli **spazi** di **lavoro**, della loro pulizia e del riordino dell'area, nonché della preparazione e manutenzione degli **utensili** di lavoro. In questo modo, con il passare del tempo, aumentarono le **capacità** e le **conoscenze** dei singoli dipendenti, rendendo più giustificato l'aumento costante del salario. Questo tipo di organizzazione, inoltre, non richiedeva la presenza di squadre specializzate addette alla pulizia, alla manutenzione dei macchinari, al controllo qualità.

Controllo qualità

A differenza di quanto si faceva nelle fabbriche statunitensi, nel modello a produzione di massa, Ōno assegnò a ogni operaio che lavorava in fabbrica, indipendentemente dal suo profilo professionale o dalla posizione all'interno dell'azienda, il compito di **controllare**, passaggio dopo passaggio, la **qualità** del prodotto man mano che questo veniva assemblato. Ogni qualvolta accadeva un problema — un accessorio da montare che non si montava correttamente, un componente che non funzionava a dovere, e così via — tutti avevano l'onere di fermarsi e provvedere immediatamente alla soluzione del problema.

Oltre a questo si sarebbero svolte immediatamente i dovuti accertamenti volti a identificare le cause del problema: in Toyota si inventò a tal proposito il cosiddetto **5Whys** [5W], il **protocollo** di **indagine** basato su 5 domande. Se durante la fase di montaggio si fosse trovato malfunzionamento, oppure un elemento danneggiato, oltre a procedere all'immediata sostituzione, tutto il team doveva interrompere il lavoro per capire la **causa** del **problema**. Ōno non si fermò a questo, ma fornì a ogni squadra un meccanismo

di stop con il quale tutti all'interno della fabbrica potevano interrompere la produzione in concomitanza di un problema evidente. Nello stabilimento furono installati i cosiddetti **tabelloni andon**, dei pannelli luminosi che riportavano in tempo reale lo stato della produzione e segnalavano in ogni istante a tutti l'insorgere di un problema, per esempio lo stop di produzione attivato da un operaio. In questo modo, tutti potevano intervenire se necessario; oppure, più semplicemente, potevano essere informati circa l'insorgere di un problema alla produzione.

Come è facile immaginare, questo schema organizzativo inizialmente rallentò in modo consistente la produzione; con il tempo però si dimostrò una intuizione vincente dato che permise di aumentare qualità e produttività, visto che i difetti e il tempo necessario per correggerli iniziarono a diminuire fino alla totale eliminazione.

Riduzione degli sprechi

Negli obiettivi di Ōno vi era quello di ridurre gli **sprechi**, che invece erano intrinsecamente presenti nel processo di produzione di massa. Come poi formulato in modo rigoroso all'interno della **teoria** della **produzione snella** (**Lean Production**), Ōno si rese conto che lo spreco poteva manifestarsi in varie forme; una di queste era causata da una **non uniforme** distribuzione del **carico** di **lavoro** all'interno della linea di produzione: in lingua giapponese si parla in questo caso di *mura*.

Nella produzione a catena di montaggio, accadeva spesso che determinati passaggi della linea di montaggio ricadessero sostanzialmente in tre categorie:

- alcuni passaggi erano sovraccarichi di lavoro, ossia si formavano i cosiddetti "colli di bottiglia";
- altri erano del tutto ingolfati, normalmente quelli a monte del collo di bottiglia;
- altri ancora, al contrario, erano troppo scarichi, senza nulla da fare, chiaramente a valle dei colli di bottiglia.

Proprio per ridurre questo tipo di spreco, Ōno decise di organizzare il personale in team completamente autonomi, in grado di svolgere tutte le fasi del processo di produzione: in questo modo si eliminava il rischio che una postazione di lavoro fosse **in attesa** di un semilavorato prodotto altrove. Grazie all'operatività e all'autonomia fornita a tutti gli operai all'interno dello stabilimento, non appena si formava un rallentamento in punto particolare del processo di produzione, altri operai avrebbero potuto accorrere ad aiutare i colleghi perché erano in grado di svolgere quel compito. Dato che a tutti fu richiesta la verifica diretta del proprio lavoro, si rese inutile la presenza di controllori e capireparto: oltre a ridurre quindi i costi diretti, questo portava i dipendenti a una maggiore **responsabilizzazione** e a un **coinvolgimento** personale nella lavorazione all'interno della fabbrica.

Obiettivi a livello di team

Un aspetto interessante legato all'organizzazione del lavoro in Toyota era quello dettato da Ōno circa il modello organizzativo delle **squadre** di lavoro sui progetti di ricerca

e sviluppo. In questo caso i team erano infatti costruiti intorno alla figura dello **Shusa**, una figura più simile al **samurai** che al caporeparto. Compito dello Shusa era quello di condurre il gruppo verso l'obiettivo proteggendo i suoi collaboratori da eventuali impedimenti e mantenendo alta l'attenzione sul progetto.

Il team veniva in genere formato da persone provenienti da reparti differenti; per esempio, per un **progetto** di una nuova auto, erano coinvolte persone dal reparto carrozzeria, da quello motoristico, da quello degli impianti elettrici e altro ancora. In tal modo, persone con specializzazioni diverse rimanevano nel team per tutta la durata del progetto, a differenza da quanto accadeva nelle aziende statunitensi; successo, carriera e onore per ogni persona del team erano legati al successo e agli **obiettivi raggiunti** dal **team** stesso. Per questo non era interesse delle persone uscire dal gruppo per tornare nei propri reparti, ma anzi tutti erano massimamente motivati a seguire in tutto e per tutto le indicazioni dello **Shusa** per far sì che il progetto andasse a buon fine.

Rapporto con i fornitori

Ōno capì fin da subito che, per migliorare la qualità del prodotto finale e al contempo ridurre rischi, costi e sprechi, era essenziale il coinvolgimento anche dei **fornitori**

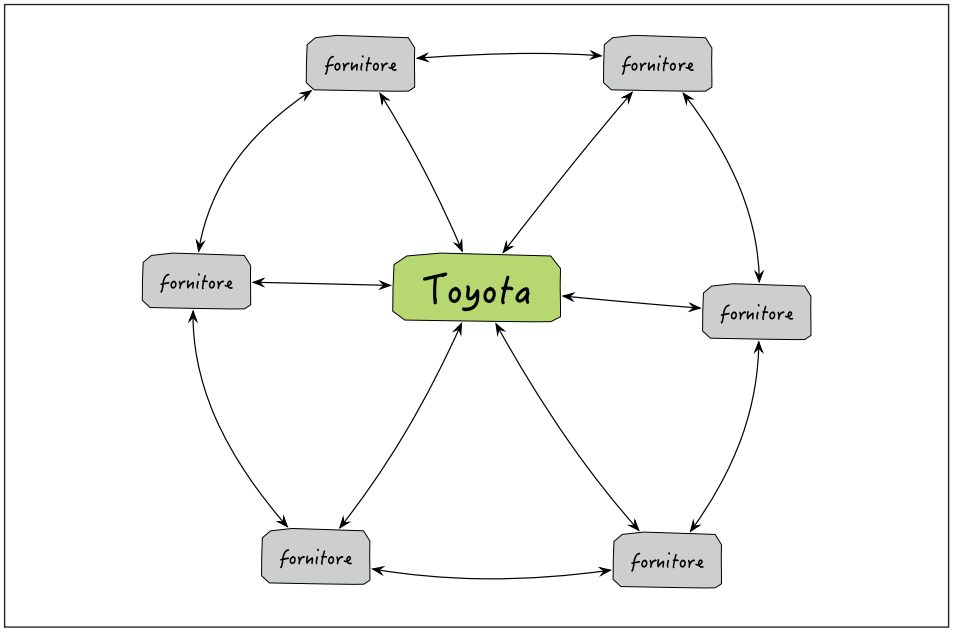


Figura I.1. Grazie alle idee di Ōno, Toyota dette vita a un network di compartecipazioni fra i vari fornitori, in cui era direttamente coinvolta. Grazie a questi accordi, i fornitori poterono pianificare meglio i loro investimenti e la produzione, mentre la casa madre poteva disporre di approvvigionamenti di miglior qualità e regolarità.

esterni nella fase di progettazione e pianificazione. I fornitori furono quindi divisi in fornitori di **primo livello** e di **secondo livello**: i primi partecipavano direttamente alla parte di progettazione proponendo quelle soluzioni tecniche che ritenevano migliori sia da un punto di vista tecnico che economico.

Toyota spingeva affinché si instaurasse un rapporto simile fra fornitori di primo e secondo livello, convinta infatti che, se il fornitore fosse stato direttamente coinvolto nella produzione delle automobili, avrebbe di volta in volta proposto la soluzione tecnicamente più valida. I fornitori a loro volta sapevano di poter contare su una collaborazione trasparente e continuativa e, in tal modo. pianificavano il lavoro senza dover dipendere da richieste improvvise o dover rincorrere il prezzo più basso per poter vincere una commessa.

Questo rapporto di reciproca fiducia era ulteriormente rafforzato da una serie di accordi societari incrociati: Toyota, alla stipula di un contratto di fornitura con un'azienda esterna, chiedeva di poter finanziare l'azienda stessa in cambio di una quota del pacchetto azionario di tale azienda; contemporaneamente, al fornitore veniva offerta una parte delle quote di Toyota. Con il tempo, i vari fornitori, convinti della bontà di questo modello di totale trasparenza e collaborazione, finirono per completare il network di compartecipazioni scambiandosi quote direttamente fra loro.

I fornitori, diversamente da quando accadeva con la Ford, non erano vincolati da rapporti di esclusiva con Toyota ma anzi erano stimolati a collaborare anche con le aziende concorrenti (Nissan, Honda, Mazda, ...): chiaramente questa politica avrebbe favorito in parte i concorrenti, i quali avrebbero ricevuto una parte del *know-how* passato da Toyota ai fornitori; ma in Toyota erano convinti che questa politica avrebbe al contempo innescato anche un flusso inverso, arricchendo i fornitori e quindi, di ritorno, Toyota stessa, con nuove soluzioni tecniche. Fu questo un primo chiaro esempio di collaborazione **win-win**, in cui tutti potevano trarre beneficio.

La storia della Toyota, terza parte: la nascita della partnership NUMMI

Il successo del modello della produzione di massa fu reso possibile grazie a una serie di fattori concomitanti, fra cui certamente l'ampia disponibilità di manodopera a basso costo. Ford fu il primo a introdurre un processo di produzione a catena di montaggio. Fu il primo a offrire alla clientela un prodotto semplice e poco costoso. Fu il primo a poter produrre un quantitativo di auto che schiacciava di fatto la concorrenza; in poco tempo divenne il principale attore sul mercato, cosa che gli permetteva di dettare le regole del gioco.

In questo contesto, il modello T per molti anni fu il modello di automobile più venduto in USA, tanto che finì per rappresentare nell'immaginario collettivo il concetto stesso di automobile: la tipologia di funzionalità, il livello qualitativo e l'affidabilità che gli acquirenti si aspettavano in un'auto erano quelli che i clienti trovavano sulla Ford T. Ford poté quindi proporre per molti anni lo stesso modello minimizzando investimenti in ricerca e sviluppo sul prodotto, concentrandosi invece sugli investimenti nel processo di produzione.

Qualche decennio più tardi, dopo il periodo di grande espansione degli anni Cinquanta e Sessanta, le condizioni che avevano reso possibile tale successo cominciarono a venir meno; iniziarono quindi a evidenziarsi in modo chiaro le **inefficienze** di questo modello organizzativo:

- costi di gestione maggiori di quelli di produzione;
- lunghi tempi di reazione per introdurre una modifica sul processo o sul prodotto;
- difficoltà nell'adeguarsi in tempi rapidi alle nuove esigenze del mercato;
- poca variabilità nel prodotto finito che invece era presentato alla clientela in poche varianti e personalizzazioni.
- In fabbrica si avevano ancora **altissimi sprechi** e una qualità molto bassa.

Contemporaneamente, in Giappone Toyota si trovava in una situazione diametralmente opposta: massima attenzione per la **qualità** del **prodotto** finito, **ridotti tempi** di **reazione** ai **cambiamenti** del mercato, focus sulle richieste del cliente e sulle dinamiche del mercato.

Toyota nel mercato americano

E a questo punto — siamo ormai alla fine degli anni Settanta — Toyota iniziò a pensare di allargare i propri interessi provando a entrare nel mercato statunitense: Eiji Toyoda e il resto del management decisero di tenere in Giappone la **progettazione** e la **produzione** delle auto, limitandosi a creare negli USA la rete di **vendita** finale. Le cose, però, non andarono proprio bene... Dopo vari tentativi più o meno fallimentari, in Toyota compresero che era necessaria una strategia differente; l'unico modo, infatti, per riuscire nell'impresa era applicare il principio incarnato dal termine giapponese **gemba** o, come si usa dire in inglese, *go & see*, ossia "vai sul posto e vedi cosa succede" ("*gemba* refers to the place where value is created").

Nel 1984 fu presa la decisione di aprire uno stabilimento Toyota direttamente negli Stati Uniti; per accelerare i tempi di realizzazione, si decise di acquistare un impianto di produzione direttamente dalla General Motors. I dirigenti giapponesi proposero l'acquisizione dello stabilimento del GM Freemont Assembly in California, che all'epoca era probabilmente il peggior stabilimento della GM, sull'orlo del fallimento: continui scioperi, tasso di assenteismo altissimo e scarsa organizzazione rendevano la produzione molto bassa. La gran parte dei lavoratori era costituita da immigrati scarsamente addestrati e ben poco motivati.

L'accordo prevedeva la creazione della New United Motor Manufacturing, Inc. (**NUMMI**), una *joint venture* fra Toyota e GM: la casa giapponese si impegnava non solo ad acquistare l'intero stabilimento e ad assumere tutti i dipendenti, ma anche a produrre una quota di modelli per la General Motors. Per questi motivi, il management della GM valutò vantaggiosa l'offerta di Toyota, anche se risultava loro abbastanza incomprensibile.

Toyota introdusse nello stabilimento di Freemont tutte le pratiche **Lean** che in svariati decenni Ōno aveva messo a punto; il risultato fu che, nell'arco di poco più di un

anno, il bilancio dello stabilimento fu portato in pareggio e subito dopo la fabbrica cominciò a guadagnare.

I manager della sede centrale della General Motors inizialmente non credettero nella performance ottenuta dalla NUMMI: in diversi pensarono che i dati fossero stati alterati, mentre alcuni arrivarono addirittura a pensare che, nottetempo, la casa madre giapponese inviasse via mare automobili già costruite in Oriente.

Quando non fu più possibile confutare il successo dell'operato dei manager giapponesi, in GM decisero di ammettere il fallimento del metodo a produzione di massa e di provare ad adottare i principi del **Lean** anche in GM. In poco tempo riuscirono a ottenere presso uno stabilimento "pilota" ottimi risultati, tanto che orgogliosamente poterono constatare un incremento del 44% dei profitti. Nel frattempo, nello stesso periodo, presso NUMMI la crescita fu del 220%.

Quello che avevano fatto in General Motors prenderà il nome negli anni successivi di **Lean di Chicago**, dal nome della città dove aveva sede il quartier generale. Gli americani avevano fatto quello che in gergo si chiama **Cargo Cult** (ne parleremo più avanti), ossia avevano messo in atto le pratiche Lean **senza averne compreso** a fondo i **principi** e soprattutto i valori. Un approccio di tale tipo vuol dire quindi perdersi per strada qualcosa di importante; vuol dire ottenere delle buone performance ma in definitiva senza capire perché ci sia stato tale incremento, e senza rendersi conto che il miglioramento potrebbe essere di portata molto maggiore.

Lo stabilimento americano di Freemont ha rappresentato un passaggio importante nella storia industriale mondiale: è qui che, in territorio straniero lontano dal proprio Paese, Toyota ha mostrato tutta l'importanza e l'efficacia della metodologia Lean. Anche quando NUMMI decise per la chiusura dello stabilimento nel 2009, questo impianto ha continuato a ricoprire un ruolo importante nei libri di storia dell'industria automobilistica. Nel maggio del 2010 infatti è stato acquistato e rimesso in produzione dalla Tesla Motors la quale, in collaborazione proprio con Toyota, progetta e produce in questo sito alcune parti della super sportiva elettrica Tesla Model S, probabilmente una delle vetture più interessanti e innovative del momento.

Un'eredità importante

La storia dell'approccio "scientifico" alla produzione industriale di massa e gli esempi relativi al mondo *automotive* ci hanno permesso di capire, da un lato, come si sia sviluppato un certo modo di concepire il *micromanagement* e, dall'altro, da quali esperienze provengano i principi *lean* e alcune pratiche che poi vedremo in opera nell'ambito dello sviluppo del software. Al di là dell'interesse culturale che questo racconto possiede, una breve introduzione storica come quella appena letta serve a ricordare che i principi Lean (e poi Agile) provengono da un lungo processo di riflessione sui metodi e sulle modalità di realizzare prodotti, e sono stati messi in pratica per decenni nell'ambito della produzione di manufatti industriali diffusi a scala globale.

Riferimenti

[TPS] Toyota Production System
http://it.wikipedia.org/wiki/Toyota_Production_System

[MCM] J.P. Wormack, D.T. Jones, D. Roos, *La macchina che ha cambiato il mondo*, Rizzoli, 1991

[TGS] Video famiglia Toyoda
indirizzo

[FHP] Highland Park Ford Plant
http://en.wikipedia.org/wiki/Highland_Park_Ford_Plant

[SM] Scientific management
http://en.wikipedia.org/wiki/Scientific_management

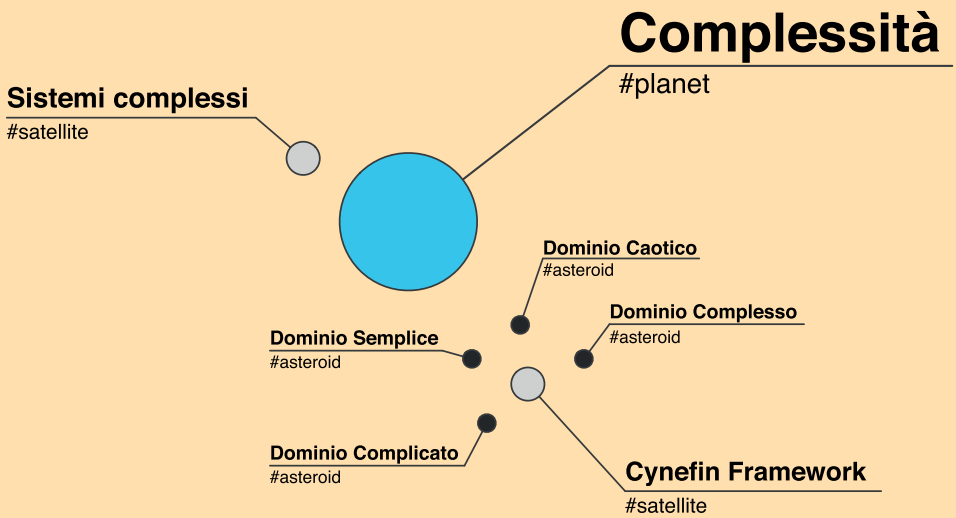
[ROU] Ford River Rouge Complex
http://en.wikipedia.org/wiki/Ford_River_Rouge_Complex

[ZAI] Zaibatsu
http://it.wikipedia.org/wiki/Zaibatsu

[KEI] Keiretsu
http://it.wikipedia.org/wiki/Keiretsu

[5W] Cinque perché
http://it.wikipedia.org/wiki/Cinque_Perché

Capitolo 2.
La complessità dei sistemi e il modello Cynefin



Un approccio globale

In questo capitolo introduciamo brevemente la tematica delle **teorie della complessità**, tramite una classificazione delle varie tipologie di sistemi e delle loro principali caratteristiche organizzative e strutturali. Attraverso l'analisi delle varie strategie da adottare nei differenti contesti, vedremo come le metodologie agili si sono dimostrate più efficaci nella gestione di progetti software tipici dello scenario IT degli ultimi anni.

È infatti fondamentale rendersi conto di come gli elementi di un progetto non possano essere ridotti meramente a tempo a disposizione, budget previsto e procedure da svolgere. Questo approccio **riduzionista**, che vede il sistema come un semplice meccanismo retto da regole rigidamente deterministiche, non tiene conto del **contesto**, delle **relazioni** fra gli elementi, delle **interazioni** fra persone e gruppi, e di numerosi altri fattori. Gli studi scientifici nel campo dell'ecologia prima, e dei gruppi sociali poi, ci hanno insegnato invece che si comprende un sistema solo se lo si guarda globalmente, sia nei suoi elementi costituenti che, soprattutto, nei rapporti e nelle interazioni che li legano.

Il modello Cynefin

Cynefin è una parola della lingua celtica gallese che significa, grosso modo, "habitat", "contesto". Il modello interpretativo che porta questo nome, messo a punto dal britannico Dave Snowden alla fine degli anni Novanta [CYN], punta a descrivere i **sistemi** sulla base dei diversi livelli di complessità; esso si basa sull'idea che ogni sistema con il quale possiamo interagire può essere classificato studiando il grado di articolazione sia del sistema stesso che dell'ambiente in cui esso si trova, in un'ottica che tiene fortemente in considerazione le interazioni reciproche.

Il **Cynefin** definisce quindi **quattro contesti** (o "domini", in inglese *domains*) di complessità: **evidente**, (in precedenza detto **semplice**) **complicato**, **complesso** e **caotico**. In figura 2 è riportata una rappresentazione grafica di tale modello; nell'immagine, le **quattro aree** sono divise da linee abbastanza nette, ma nella realtà la separazione non è così marcata: un sistema quindi non si trova quasi mai in uno dei quattro quadranti in modo netto, ma sono sempre presenti delle **sfumature** che possono variare da caso a caso.

Inoltre, variando il punto di osservazione o la scala con la quale lo si prende in esame, un sistema può essere classificato di un tipo piuttosto che di un altro. Una colonia di insetti sociali, se presa nella sua interezza, è un sistema semplice, al massimo complicato. Se invece si osservano le dinamiche che regolano la struttura sociale interna di tale colonia, allora si può certamente parlare di un sistema complesso.

Un modello per interpretare i sistemi

Il **Cynefin** è uno **strumento teorico** di classificazione che definisce sia quali siano le dinamiche comportamentali dei sistemi stessi, sia quali possano essere le strategie più adeguate a seconda del caso: per risolvere un problema complicato servono gli esperti? E per un problema complesso? Si possono applicare positivamente a una situazione

delle soluzioni utilizzate in passato e già sperimentate in altri casi? Causa ed effetto sono sempre direttamente collegabili?

Di seguito presentiamo i diversi domini previsti da questo modello interpretativo; nella lingua corrente purtroppo spesso si confonde il significato delle parole **complicato** e **complesso**, che invece sono due cose diverse fra loro. Lo stesso termine "complesso" può essere usato sia per descrivere il tema della complessità sia per indicare un sistema complesso. D'ora in poi useremo il termine "complessità" per indicare genericamente l'argomento trattato nel campo delle "teorie della complessità", una branca delle scienze cognitive che attinge al mondo della filosofia, dell'ecologia e delle discipline sociali.

Diverso è il discorso per la parola "complesso", che indica un caso specifico quando si descrivono i quattro domini del modello Cynefin ordinati dal più semplice al più articolato: **semplice** (**evidente**), **complicato**, **complesso**, **caotico**.

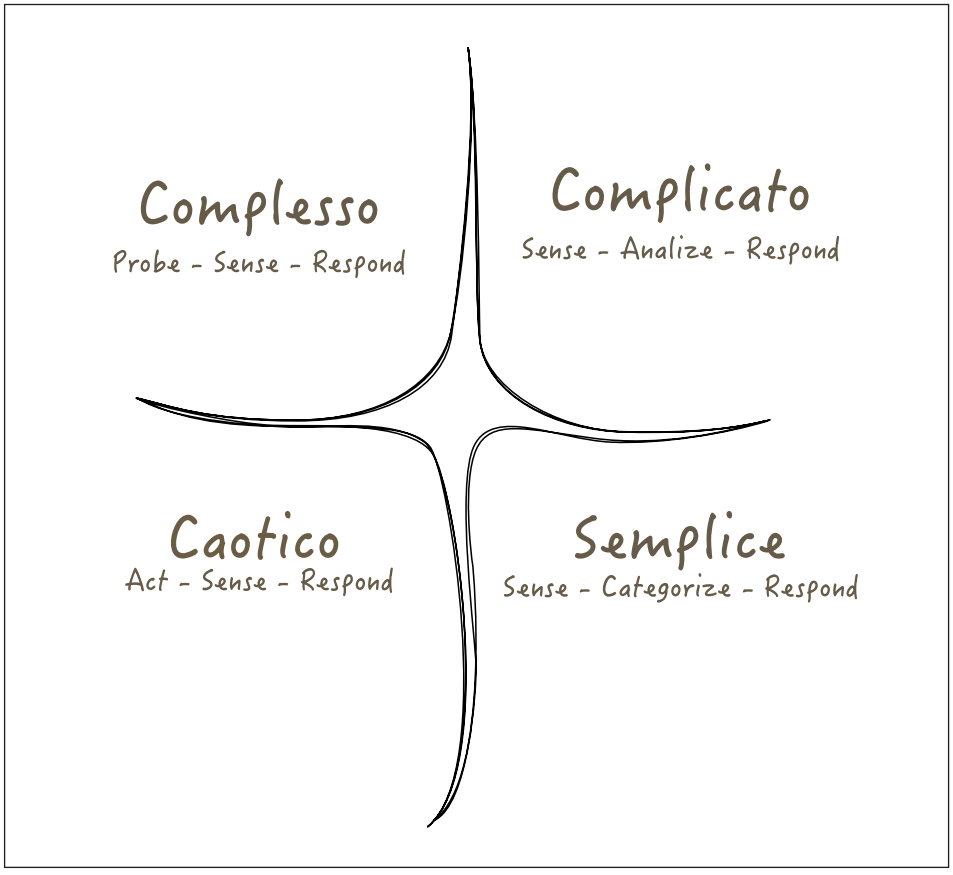


Figura I.2. I domini di complessità di Cynefin. Dal basso a destra, in senso antiorario, troviamo in sequenza il dominio semplice, quello complicato, quello complesso e infine quello caotico.

Dominio semplice (evidente): sense-categorize-respond

Il primo quadrante è quello del dominio **evidente**, precedentemente detto **semplice**, quello a complessità minore; in questo contesto, **causa** ed **effetto** sono **strettamente correlati** e ogni azione porta a una conseguenza facilmente prevedibile: "se rovescio un bicchiere d'acqua sulla tovaglia, questa si bagna". L'azione è ripetibile e otterremo sempre lo stesso risultato finale; il rapporto tra causa ed effetto è chiaro ed è facilmente comprensibile da chiunque: non è necessaria una competenza specifica. Da notare che analizzare il moto percolatorio delle particelle d'acqua all'interno del tessuto della tovaglia invece è un problema tutt'altro che semplice, che probabilmente segue la teoria dei moti caotici...

In questo caso la variabilità è limitata, le **possibili soluzioni** sono poche e facilmente **individuabili**: in molti casi, la soluzione al problema è semplicemente una, è evidente a tutti e non è in discussione.

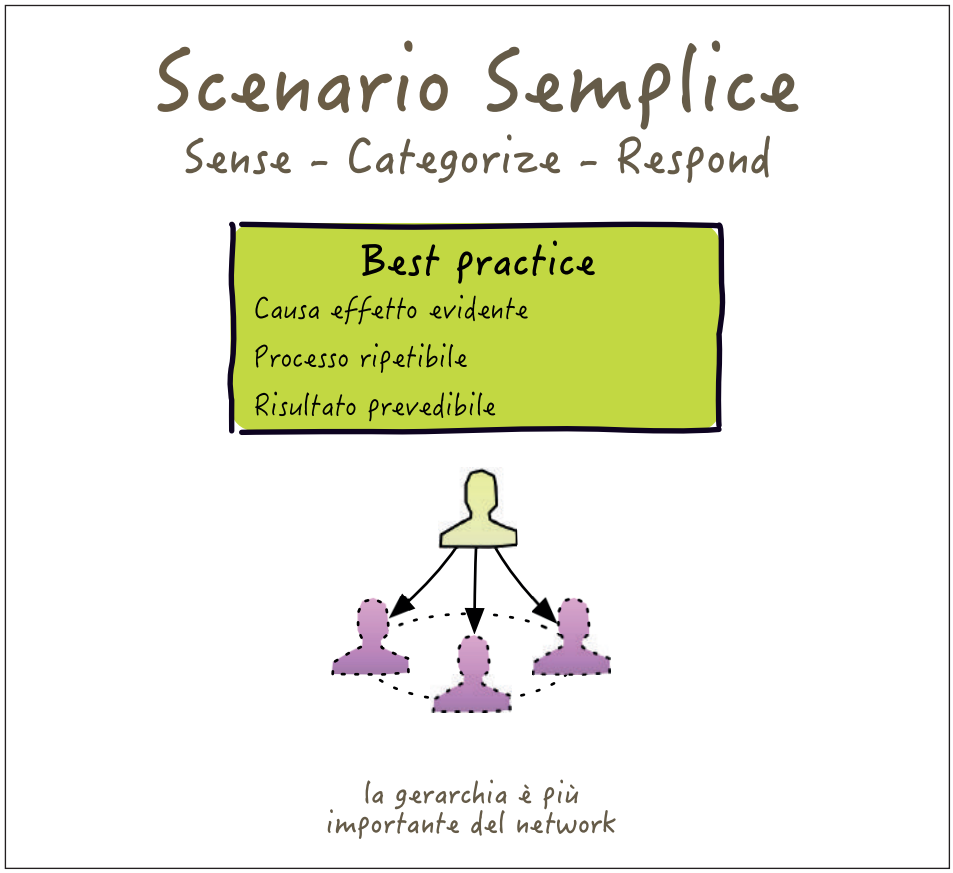


Figura I.3. Un sistema semplice secondo il modello Cynefin. Si agisce tramite l'applicazione delle best practice, usando il pattern sense-categorize-respond e il fatto che la comunicazione è più efficace quando segue un modello gerarchico.

Siamo quindi nel contesto detto dei *known knowns*, frase non direttamente traducibile in italiano, ma il cui senso potrebbe essere "**sappiamo le cose che sono da sapere**": le informazioni sono disponibili oppure è possibile ottenerle facilmente. In una situazione semplice, i cosiddetti *decision makers* mettono in pratica un processo del tipo **sense–categorize–respond**: valutare, classificare e agire di conseguenza.

Fra i contesti **semplici**, si possono trovare per esempio le classiche attività di processo standardizzato di produzione o di assemblaggio, come la classica catena di montaggio inventata da Ford. In un contesto semplice, probabilmente la maggior parte dei problemi sono stati già affrontati e risolti in passato, da noi o da altri: per questo l'utilizzo di *best practices* è in questo caso una pratica consolidata e la reingegnerizzazione è uno strumento di uso comune.

Sia chi coordina che chi esegue ha accesso alle informazioni e lo stile *command–and–control* è quello tipicamente preferibile. In questo caso la **gerarchia** è importante, probabilmente prevalente sul *network*: in questo caso infatti ogni azione può essere intrapresa con poche informazioni, per cui è più efficiente seguire una direttiva che arriva dall'alto, piuttosto che intavolare una discussione fra i diversi elementi coinvolti (il *network*, la **rete** di relazioni tra i vari attori) per stabilire quale sia il modo migliore di agire.

Dominio complicato: sense–analyze–respond

Aumentando il livello di complessità si trova il dominio **complicato**, noto anche come il dominio **degli esperti** dove la strategia vincente consiste nell'affidarsi al *know-how* degli esperti; in questo ambito infatti non esiste una sola soluzione al problema, ma sono possibili molteplici **alternative**.

La differenza rispetto al caso precedente è data dal fatto che la relazione fra causa ed effetto, benché ancora presente, non è così evidente a tutti; la comprensione di tale relazione dipende dal livello di conoscenza ed esperienza dei vari *decision maker*. L'analisi della relazione causa–effetto è efficace solo in funzione della conoscenza che l'esperto ha del dominio in questione.

In un dominio **complicato**, il numero di variabili, di incertezze e di perturbazioni aumenta rispetto al dominio semplice, facendo crescere l'articolazione dei problemi; aumenta pertanto il numero delle possibili risposte. Contrariamente al dominio semplice, qui **sappiamo di non sapere**, ma sappiamo quali sono le domande da fare; sappiamo che è necessario consultare gli esperti che possono trovare tali risposte.

Spesso l'approccio preferito dagli esperti per risolvere un problema è quello dell'**analisi**: il problema viene scomposto nelle sue parti costituenti, in modo da procedere alla soluzione dei sotto-problemi, che in genere sono quindi più maneggevoli; si assume quindi che la **somma delle soluzioni** possa portare alla soluzione complessiva.

In un dominio complicato, questo approccio di **scomposizione** è efficace proprio perché i singoli componenti hanno una **bassa interdipendenza**: benché tutte le parti lavorino insieme all'interno dello stesso sistema, il loro funzionamento e comportamento è relativamente indipendente da quello delle altre componenti. Il funzionamento del

singolo elemento è invariante sia se preso singolarmente che quando inserito nel sistema: alterando il funzionamento di uno dei componenti del sistema non si hanno ripercussioni significative sul funzionamento degli altri.

La caratteristica di un sistema **complicato** di essere compatibile con un processo di scomposizione e ricomposizione ha dato vita in passato anche a qualche celebre errore di valutazione. In molti iniziarono a pensare che il processo di scomposizione si potesse applicare a tutti i problemi o alle casistiche più disparate, senza invece considerare il tipo di articolazione del sistema. Come conseguenza di questa convinzione, si consolidò l'idea errata che un qualsiasi sistema produttivo potesse essere ottimizzato tramite l'ottimizzazione di ciascuna sua parte. Questa corrente di pensiero, che prende il nome di **riduzionismo**, tende a sottovalutare il livello di complessità di un sistema. Nella prima metà del Novecento, furono formulate numerose teorie spesso basate però su assunti inesatti. Un esempio tipico è legato a una superficiale interpretazione del pensiero di

Figura I.4. Il dominio complicato: in questo caso agire per best practice è la soluzione migliore. La comunicazione dall'alto è utile come anche quella del network.

Deming; egli fu, fra le altre cose, l'autore delle **14 regole sulla qualità totale** [14QT], molto citate nella letteratura lean e nelle migrazioni agili.

Un'errata idea di "ottimizzazione"

Quando Deming affrontò, nei suoi studi, il tema della scomposizione e ricomposizione, in molti si affrettarono a usare le sue teorie come giustificazione per l'implementazione di costose operazioni di ottimizzazione dei processo di produzione. È probabilmente da una sua celebre frase che questo movimento di pensiero si è formato: "Nell'azienda, mettere tutti al lavoro per realizzare la trasformazione. La trasformazione è il lavoro di tutti".

In questa frase, con il termine "trasformazione" si intende il processo stesso di ottimizzazione all'interno dell'organizzazione. L'errore è legato alla sbagliata interpretazione della parola "tutti", secondo la quale il miglioramento del sistema era ottenibile tramite l'ottimizzazione di tutti gli attori coinvolti. Sebbene più tardi Deming la smentisse, questa interpretazione per molto tempo ha continuato a propagarsi nel mondo del management.

In una sua celebre pubblicazione [DEM] egli infatti afferma: "L'ottimizzazione è un processo di orchestrazione degli sforzi di tutte le componenti verso il raggiungimento dell'obiettivo stabilito. L'ottimizzazione è il lavoro del management. Tutti vincono, con l'ottimizzazione. Tutto ciò che non punta all'ottimizzazione dell'intero sistema comporterà una perdita finale per ogni componente del sistema. Ogni gruppo dovrebbe porsi come obiettivo l'ottimizzazione nel lungo periodo del sistema più ampio in cui esso opera. L'obbligo di ciascuna componente è di contribuire al meglio al sistema, non di massimizzare la propria produzione, il proprio profitto, le proprie vendite o qualsiasi altra misura della competitività. **Alcune componenti potrebbero operare 'in perdita' al fine di ottimizzare il sistema nel suo insieme, comprese appunto quelle componenti che vanno in perdita**".

Quindi Deming era convinto che spingere al massimo tutte le parti di un sistema non fosse il modo più efficace per ottenere il risultato migliore dal sistema: far lavorare a regimi più bassi alcune componenti, che siano semplici macchinari, comparti di lavorazione o persone, spesso permette di ottenere prestazioni complessivamente migliori.

Tale errata valutazione portò alla nascita di un gran numero di strumenti il cui obiettivo era **migliorare** le **singole** componenti di processo, ma ben poche metodologie per un **miglioramento** del sistema **complessivo**.

Concentrare tutte le energie nella ricerca di metodi di miglioramento della performance e della riduzione dei costi fece inoltre perdere di vista gli obiettivi primari di questa politica di gestione. Se da un lato infatti si poterono individuare alcuni errori grossolani, dall'altro si introdussero gradualmente sistemi di gestione via via più complicati; le organizzazioni videro quindi nascere nuovi e ulteriori costi necessari per sostenere i maggiori impegni organizzativi e di gestione. Ben presto si arrivò alla paradossale situazione per cui i **costi** di **gestione** cominciarono a pareggiare se non superare quelli legati alla produzione.

Come avremo modo di vedere nel prossimo paragrafo quando parleremo di sistemi complessi, uno dei fattori o, per meglio dire, degli errori che ha influito nel trasformare il processo di riduzione dei costi in qualcosa di diametralmente opposto, è stato quello di ignorare la (crescente) complessità dei sistemi, e quindi continuare ad applicare schemi e metodi validi in passato ma che stavano mostrando tutti i loro limiti.

Dominio complesso: probe-sense-respond

Arriviamo ora a cercare di comprendere in cosa consista il dominio del **complesso**; nell'uso comune della lingua italiana complicato e complesso sono spesso confusi l'uno per l'altro, ma non è corretto. Grosso modo possiamo dire che un sistema complesso è ancor più articolato di uno "solamente" complicato; ma la vera differenza tra un sistema complicato e uno complesso sta nell'organizzazione del sistema sia per quanto riguarda la **struttura** (visione **statica**) che il **comportamento** (visione **dinamica**).

La prima definizione che può soddisfare un pubblico di matematici è quella che un sistema complesso **non è modellabile** per mezzo di equazioni lineari: definizione interessante, da sfoggiare in una serata con amici che hanno studiato fisica o matematica, ma che lascerebbe piuttosto indifferenti tutti gli altri...

Feedback rientrante

Proseguendo possiamo dire che un sistema complesso è **rientrante** ossia è dotato di caratteristiche di **retroattività cortocircuitata**: il risultato del processo di lavorazione rientra in parte nel circuito stesso, influenzandone il comportamento (**feedback**).

Un esempio tipico di sistema rientrante è il corpo di un atleta durante una prova di resistenza: in funzione del tipo di allenamento e di resistenza, esiste una soglia (detta soglia aerobica) oltre la quale i muscoli iniziano inevitabilmente a produrre acido lattico. Più l'atleta "spinge" oltre tale soglia, più produrrà una quantità sempre maggiore di acido lattico, che di fatto riduce l'efficienza delle cellule muscolari, compromettendo l'efficacia dello sforzo. Se l'abbassamento della prestazione non è un risultato ammesso, magari quando in ballo vi è un traguardo importante, l'atleta "spinge" ulteriormente per compensare questo calo; e questo non fa altro che aumentare la produzione di acido lattico che nuovamente influenza negativamente la prestazione.

Si potrebbe dire che si è innescato in questo caso un **cortocircuito** con trend negativo. Esistono anche casi opposti in cui il feedback rientrante permette di aumentare la prestazione del sistema, si pensi alla marmitta catalitica o al classico effetto Larsen: quest'ultimo è il classico fischio che si verifica quando si avvicina una chitarra elettrica all'altoparlante dell'amplificatore cui essa è collegato: se suonate in una band di *noise* giapponese, però, probabilmente il rumore distorto provocato dal feedback rientrante del vostro strumento farà esaltare i fan più esagitati sotto il palco...

Passando a un esempio più collegato alla realtà IT, si pensi al caso di un team che si trovi già in una situazione svantaggiata, dovendo compiere un lavoro incompatibile con le proprie possibilità: tempi stretti, gruppo di lavoro male assortito, requisiti non chiari. In un

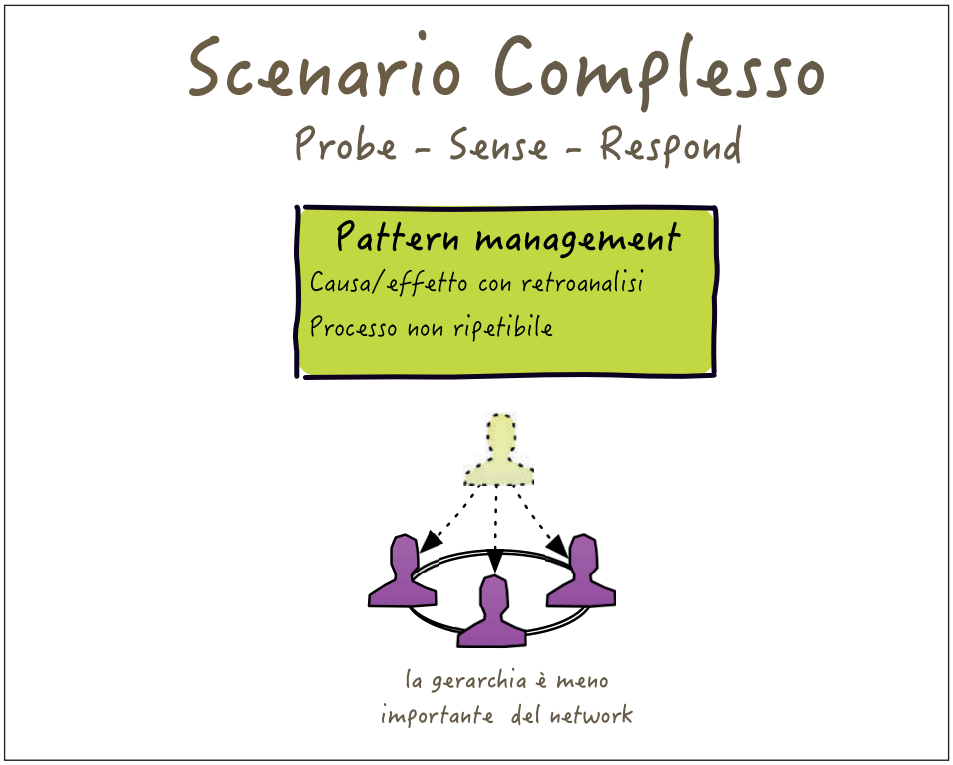


Figura I.5. Il sistema complesso: le best practices non sono più utilizzabili. La comunicazione del network è di gran lunga più utile di quella che viene dall'alto.

tale contesto, la pressione che naturalmente si genera sulle persone del team non fa altro che peggiorare la situazione, riducendo per esempio le prestazioni e creando spesso ritardi nelle consegne o abbassando la qualità del prodotto. Ritardi e bug porteranno nuovamente un carico di stress sulle persone che peggioreranno ulteriormente le loro prestazioni.

Adattività e interconessione

Strettamente connessa al concetto di feedback rientrante, un'altra caratteristica tipica dei sistemi complessi è quella dell'**adattività**: introducendo infatti in un sistema degli elementi di disturbo, con lo scopo di modificarne il comportamento, si ottiene spesso una reazione del sistema tale da annullare l'intervento stesso. Un esempio di questo comportamento è quello tenuto dalla borsa valori; supponendo, per esempio, di disporre di una qualche formula matematica o modello di previsione che ci permettesse di guadagnare sul trend di un qualche titolo, il fatto stesso di investire sulla base delle informazioni forniteci dal modello in nostro possesso introdurrebbe un'alterazione nel sistema tale da annullare totalmente l'efficacia degli strumenti in nostro possesso.

Un sistema complesso è in genere governato dalla **serie storica**, ossia il comportamento di oggi e le risposte di domani sono funzione di quello che è successo dentro e fuori dal sistema fino a ieri.

Da un punto di vista strutturale un sistema complesso si caratterizza per essere composto da **parti** fortemente **interconnesse** fra loro: il funzionamento di ogni singola componente è strettamente legato e, per certi versi, **dipende**, da quello delle parti vicine. In tal senso i social network, le folle di persone accalcate ad ascoltare concerti di *noise* giapponese, le colonie di insetti sociali, il traffico cittadino e gli adepti delle diete improbabili sono sistemi complessi.

Le parti e il tutto

Nel 1978 Arthur Koestler coniò termine **olone** [OLO], per definire un componente di un sistema, dotato di una propria individualità ma che, unendosi ad altri elementi simili, si integra in un sistema di ordine superiore. Gli oloni sono in genere elementi **frattali**, ossia sono composti da altri sottosistemi, che solitamente sono degli oloni anch'essi.

Un sistema **olistico** è quindi un insieme o gruppo di parti interdipendenti o temporalmente collegate: queste parti sono a loro volta sistemi e sono composte da altre parti, in uno schema frattale. Secondo la teoria dei sistemi, il modo per comprendere a fondo il comportamento dei sistemi olistici sta nel valutarli nella loro **interezza**, come per il caso degli insetti sociali, dove la colonia si comporta come un superorganismo unico. Analogamente, la teoria del *systems thinking* pone l'attenzione sul concetto di **interconnessione** fra elementi.

Un **sistema complesso** è quindi un particolare tipo di sistema olistico, dove viene a mancare l'aspetto **ricorsivo–frattale**. L'esempio più chiaro per spiegare tali differenze è quello offerto dagli insetti sociali: una colonia di api è composta da un **insieme** di **oloni** che si comportano come un tutt'uno. Però, due colonie non si aggregano per formarne altre e quindi manca la frattalità verso l'alto; al tempo stesso, gli oloni fondanti, cioè le singole api, non sono il risultato dell'aggregazione di altri oloni.

Sistemi complessi: comportamento e strategie di gestione

Il comportamento del **sistema complesso** è dato dalla somma dei comportamenti dei singoli componenti i quali interagiscono fra loro per **adattarsi** o **apprendere**: essi sono in grado di osservare l'impatto delle loro iniziative e di regolarle in maniera adeguata al fine di raggiungere il risultato voluto a livello di sistema. In questo senso potremmo quindi dire che i sistemi complessi sono le "organizzazioni in grado di apprendere" descritte da Peter Senge nel suo famoso libro *La quinta disciplina* [PS].

Per questi e altri motivi, **non** è possibile **isolare** il comportamento dei singoli componenti di un sistema complesso: se presi singolarmente infatti possono o smettere di funzionare o agire in modo completamente differente. Osservare il **singolo** isolato fornisce un'informazione molto limitata su quello che sarà il comportamento all'interno del sistema nel suo complesso.

Dopo aver visto quindi alcune caratteristiche **strutturali** e **comportamentali** di un sistema complesso, può essere utile tentare di delineare le strategie di gestione per un sistema di questo tipo. A differenza del dominio complicato, dove valgono l'esperienza e l'applicazioni delle *best practices*, in questo caso pianificazione preventiva e classificazione per ricondurre a qualcosa di noto sono pratiche poco utili: troppe sono le variabili che possono modificare il comportamento del sistema. La conoscenza degli esperti è meno importante per una valutazione causa–effetto, o comunque può funzionare solo in ambiti marginali. Invece di osservare il sistema dall'esterno per provare a ipotizzare una strategia di intervento, in questo caso è più efficace **sperimentare direttamente** sul campo le strategie e vederne immediatamente gli effetti e l'efficacia.

Lavorare in un dominio complesso richiede competenze ed esperienze su aspetti diversi, che difficilmente si possono trovare in una sola persona: serve quindi un team multidisciplinare con competenze ampie e distribuite. Per questo motivo la gerarchia è meno utile del network: più che il parere di un singolo, qui vale l'**interazione** di persone esperte in ambiti differenti; funziona il lavoro di squadra piuttosto che seguire gli ordini di un leader. In tal senso Peter Dreker, autore di fama mondiale per le sue opere sulle teorie di gestione aziendale [PD], ha affermato che: "il lavoratore ha competenze dettagliate sul dominio in cui lavora rispetto al suo capo; il capo non ne capisce a sufficienza per comandarlo e controllarlo adeguatamente".

Dominio caotico: act–sense–respond

Il legame che sussiste fra causa ed effetto può rappresentare uno strumento di valutazione e indagine particolarmente utile nel caso semplice e complicato. Anche nel modello complesso può comunque fornire informazioni utili, anche se si deve tener conto che la conoscenza acquisita spesso ha una validità limitata nel tempo e che quindi sono necessari in questo caso nuovi cicli di indagine e di raccolta di informazioni.

Lo scenario turbolento, tipico di un dominio **caotico**, è caratterizzato da elevato grado di incertezza. Il comportamento di un sistema caotico però non è casuale, ma segue leggi tipiche della teoria del caos, in cui una minima variazione delle condizioni di partenza può portare a risultati completamente differenti da quelli sperimentati un istante precedente. La celebre metafora formulata dal matematico statunitense Edward Lorenz agli inizi degli anni Settanta, e nota come *The Butterfly Effect* descrive esattamente questo comportamento: "se oggi una farfalla sbatte le ali in Africa, domani pioverà a New York".

Anche se caotico è diverso da casuale, da un punto di vista pratico, un sistema caotico è comunque **impredicibile**; questo è il dominio dell'ignoto e, probabilmente, anche del non conoscibile (*unknown unknowns*): non solo non si conoscono le risposte, ma nemmeno si sanno le domande.

L'approccio alla gestione di un sistema caotico

Quando però ci troviamo nel bel mezzo di una emergenza, probabilmente la sola cosa utile da fare è agire. Per questo motivo questo scenario viene detto **superhero domain**:

piuttosto che stare a discutere sulle decisioni da prendere, in questo caso può essere più utile seguire un leader "supereroe" senza farsi tante domande.

Purtroppo spesso accade che non ci sia un leader da seguire, oppure che il leader storico, e magari molto capace, sia ormai stanco di dover risolvere l'emergenza a rischio della propria salute. In ambito sviluppo del software il caos si può manifestare in svariati modi; tipicamente prende la forma di un progetto totalmente fuori controllo: scadenze completamente saltate, un cliente arrabbiato per il ritardo nelle consegne o per

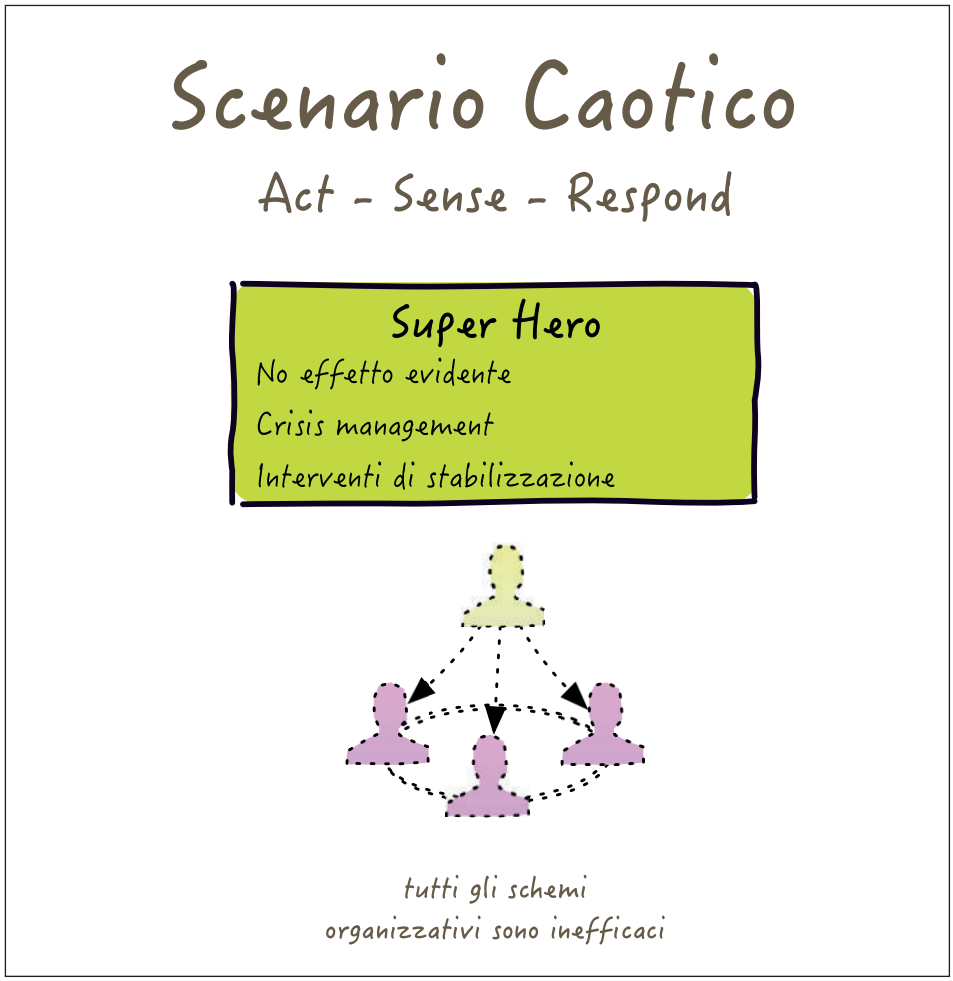


Figura I.6. Quando però ci troviamo nel bel mezzo di una emergenza, probabilmente la sola cosa utile da fare è agire. Per questo motivo questo scenario viene detto super hero domain: piuttosto che stare a discutere sulle decisioni da prendere, in questo caso può essere più utile seguire un leader senza far tante domande.

la scarsa qualità del software rilasciato, un team che lavora a ritmi insostenibili per lungo tempo o altro ancora.

In una situazione come questa inizia a serpeggiare l'idea che il fallimento sia dietro l'angolo. Nel mondo dello sviluppo del software spesso si usa la metafora della cosiddetta *rock star*, un team leader molto esperto, con una conoscenza tecnica superiore a quella dei colleghi. Alla *rock star* normalmente si affida il ruolo del supereroe in grado di portare il progetto a conclusione. A volte ci riesce, a volte no. A volte scappa, ossia si licenzia per un lavoro più tranquillo che gli lasci tanto tempo libero per poter assistere a concerti di *noise* giapponese. A volte finisce per assumere il totale controllo del vascello alla deriva nel mare tempestoso.

È importante tener presente che, qualora la gestione del progetto dovesse entrare in uno stato caotico, dovremo mettere in conto un'alta probabilità di fallimento. In questi casi, la cosa più conveniente da fare, sia dal punto di vista dell'economia di progetto ma anche della gestione del gruppo di lavoro e dell'azienda, è pensare se sia il caso di abortire il progetto; non sempre è possibile.

Qual è la complessità di un sistema?

Dopo aver analizzato i differenti tipi di complessità, alcune domande sorgono spontanee sulla natura dei sistemi all'interno dei quali ci muoviamo: un sistema complicato o uno complesso lo sono per loro natura o semplicemente per una nostra incapacità di vedere le cose in modo univoco? Possiamo far nulla per ridurre la complessità di un sistema? Un sistema caotico può essere ricondotto a qualcosa di diverso?

Rispondere a queste domande non è immediato. Certamente ci sono sistemi che sono **per loro natura** complessi: il tempo atmosferico, le organizzazioni composte da persone, alcune aggregazioni di insetti, gli ecosistemi e così via. Ogni tentativo di analizzarli in maniera riduzionista porta al massimo a dei **modelli statistici**, più o meno validi ma essenzialmente incompleti.

Altri sistemi **diventano complessi** perché crescono e non sono più compresi nel loro insieme totale. Un esempio piuttosto chiaro potrebbe essere quello di un prodotto software: fintantoché il prodotto è piccolo, potremmo considerarlo semplice; se le dimensioni e le interdipendenze aumentano, probabilmente dovremo considerarlo prima complicato e poi complesso. In questo senso il Cynefin è un sistema di **sensemaking**: se definiamo un sistema semplice, complicato o complesso dipende dal tipo di conoscenza che abbiamo o possiamo avere su tale sistema; detto in altri termini, dipende da quanto siamo in grado di creare un modello capace di predirne la futura evoluzione.

Un'organizzazione sociale di esseri umani è essenzialmente un sistema complesso per quanto ne capiamo al momento. Saremo mai in grado di creare un modello che ne predica il futuro? Al momento sembra improbabile.

Una delle conseguenze di un *sensemaking* tramite Cynefin è il fatto che, nel momento in cui classifichiamo un sistema come complesso, abbiamo una classe ben precisa

di interventi a disposizione: **probe**, **sense**, **respond**, ossia **sondare** il sistema, **valutare** quello che emerge dal nostro "sondaggio" e **rispondere** in maniera adattiva a quello che possiamo comprendere.

In definitiva la strategia più saggia è quella di creare esperimenti controllati e capire quali ci aiutano ad alterare il sistema nel modo voluto. Dave Snowden parla di esperimenti **safe-fail**, cioè di lavorare in contesti controllati e in ambiti dimensionalmente controllati (dal punto di vista temporale o di dominio) in modo che l'eventuale fallimento non metta a rischio l'intera organizzazione o l'intero progetto.

Riferimenti

[CYN] D.J. Snowden, *Cynefin: a sense of time and space, the social ecology of knowledge management*, in C. Despres, D. Chauvel, *Knowledge Horizons: The Present and the Promise of Knowledge Management*, Butterworth-Heinemann, 2000

[PD] Scheda di Peter Drucker su Wikipedia
http://it.wikipedia.org/wiki/Peter_Drucker

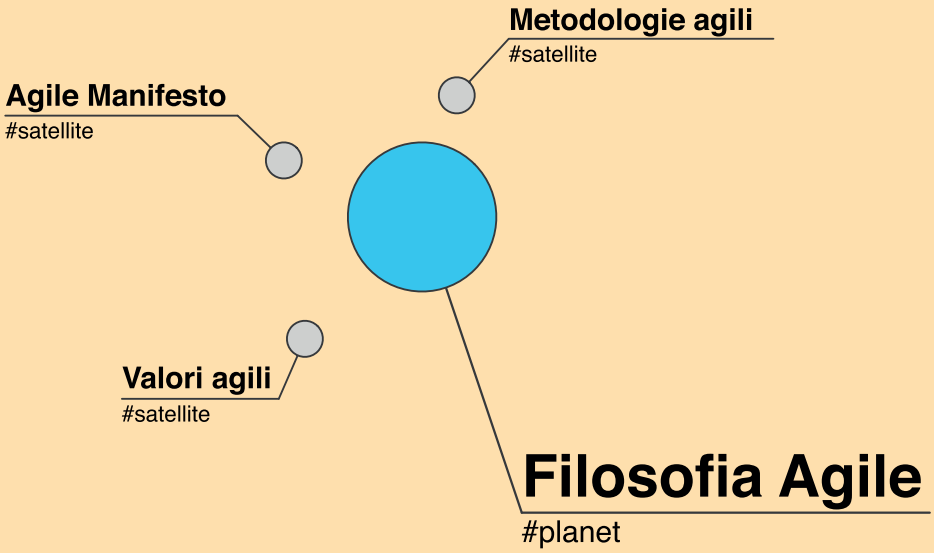
[OLO] La voce "Olone" su Wikipedia
http://it.wikipedia.org/wiki/Olone

[PS] P. Senge, *La quinta disciplina: l'arte e la pratica dell'apprendimento organizzativo*, Sperling & Kupfer, 2006

[14QT] Le 14 regole della qualità totale secondo Deming
http://goo.gl/hMM1c

[DEM] W.E. Deming, *The New Economics for Industry, Government, Education*, (2nd ed.), The MIT Press, 2000

Capitolo 3
Filosofia Agile



Che cosa sono le metodologie agili

Il "senso comune" (che non necessariamente coincide con il "buon senso") ci ha abituato al seguente modo di affrontare lo svolgimento di una qualsiasi operazione: per portare a termine il compito prima si procede a una analisi completa delle cose da fare, poi si realizza una pianificazione dettagliata in tutti gli aspetti e infine ci si mette a svolgere in sequenza ogni singola operazione fino al completamento. Si parla a volte di organizzazione orizzontale delle attività: per esempio per restaurare una casa prima si fanno (tutti) i progetti, poi si procede a eseguire (tutti) i lavori di rifacimento delle fondamenta, poi si passa a (tutti) i lavori sulle mura, fino ad arrivare alla imbiancatura di (tutte) le stanze. È un approccio che, in determinate situazioni, ha le sue ragioni e si è dimostrato funzionante.

I problemi nascono quando si pretende di applicare tale paradigma ad attività a cui invece mal si adatta questo modo di lavorare. Le caratteristiche tipiche di un sistema **complesso**, ad esempio, suggeriscono che sia necessario un approccio diverso da quello a cui tipicamente siamo abituati: pianificare una strategia prima di essere "entrati nel problema" potrebbe non essere di alcun aiuto; il fatto stesso di suddividere il lavoro in attività orizzontali potrebbe non avere alcun beneficio.

Quello che la teoria della complessità ci suggerisce è di "entrare" nel caso, vedere che succede, provare a risolvere qualche problema (scomposizione verticale) e capire se quello che stiamo facendo sia efficace o meno.

Le **metodologie agili**, partendo da questi presupposti, si preoccupano di definire un metodo di lavoro che sia efficace nei contesti complessi, quali sono ad esempio quelli dello sviluppo e della produzione di software o della cosiddetta "economia della conoscenza". Concetti come lavoro e collaborazione del gruppo, iterazioni piccole, *fail-fast / fail-safe*, *dance with the system*, visualizzazione dei flussi e altro ancora sono infatti colonne portanti delle metodologie agili.

Qualche cenno storico

Il termine **metodologia agile** fu coniato nel 2001 quando un gruppo di esperti, consulenti e progettisti dell'IT, che rappresentano i *guru* di buona parte dell'informatica moderna, si riunì per discutere su alcuni problemi tipici dell'**ingegneria del software**. Fu identificata una serie di principi e valori ritenuti importanti per migliorare il modo di lavorare e di produrre il software. Il lavoro di queste persone finì per dar vita a quello che oggi conosciamo come il **Manifesto Agile** [MA] e per far partire il cosiddetto **Agile Movement**.

Il termine inglese *agile*, di significato sovrapponibile all'analogo termine italiano, è stato scelto per esprimere un modo di intendere l'ingegneria del software in contrapposizione con i modelli precedenti; l'evoluzione delle metodologie infatti parte dalle cosiddette **metodologie predittive** (o basate sul modello "a cascata", *waterfall*), passando per le **metodologie iterative** (modello a spirale, RUP – *Rational Unified Process*) arrivando infine alle **metodologie agili**.

Il celebre **Agile Manifesto** è organizzato sulla base di quattro **valori agili** (che rappresentano il cuore del manifesto stesso) e su una serie di **principi agili** i quali dettagliano meglio i concetti, derivando direttamente dai valori.

Valori agili

Di seguito è riportata la traduzione in italiano della prima pagina del *Manifesto per lo sviluppo agile di software* che enuncia i quattro valori fondamentali:

> Stiamo scoprendo modi migliori di creare software, sviluppandolo e aiutando gli altri a fare lo stesso. Grazie a questa attività siamo arrivati a considerare importanti:
>
> | | | |
> |---|---|---|
> | **Gli individui e le interazioni** | *più che* | i processi e gli strumenti |
> | **Il software funzionante** | *più che* | la documentazione esaustiva |
> | **La collaborazione col cliente** | *più che* | la negoziazione dei contratti |
> | **Rispondere al cambiamento** | *più che* | seguire un piano |
>
> Fermo restando il valore delle voci a destra, consideriamo più importanti le voci a sinistra

Questi punti furono proposti come i pilastri su cui costruire il lavoro per risolvere i problemi fino ad allora riscontrati nel mondo dell'ingegneria del software. Appare chiara fin da subito l'importanza del **focus** sulle **persone**; forse può risultare meno scontato il concetto di **valore**: come avremo modo di vedere, per gli "agilisti" produrre valore significa realizzare un prodotto che **soddisfi le necessità** dell'utente finale. Per questo parole come **prodotto** e **necessità di business**, sostituiscono progetto e feature: l'utente non ha alcun interesse che un progetto sia pianificato e completato, mentre è interessato alla realizzazione del prodotto che gli permetta di lavorare o di svolgere una qualche attività.

Il manifesto ci dice anche che, per rilasciare valore, è indispensabile la **collaborazione con il cliente**, cosa che si ricollega con quanto detto nel capitolo precedente quando si è parlato di scenari complessi: in questi contesti infatti solo stando sul campo e "vivendo" la quotidianità si possono conoscere i bisogni degli utilizzatori. Per questo motivo il team e l'utente finale possono, meglio di chiunque altro, individuare le necessità e trovare i modi per soddisfarle.

Il cambiamento ha un costo, viva il cambiamento

Ultimo punto, ma non per questo meno importante, è quello in cui si pone l'attenzione sul saper gestire e rispondere al **cambiamento**.

Il cambiamento è parte integrante in un progetto software: arrivano nuove richieste in corso d'opera, cambia l'interpretazione di alcuni requisiti, così come nuove informazioni che emergono durante il ciclo di lavorazione permettono di comprendere meglio alcuni passaggi, contribuendo a dare una nuova rotta allo sviluppo. Chi ha lavorato in un progetto software sa quanto spesso tutti questi eventi possono verificarsi.

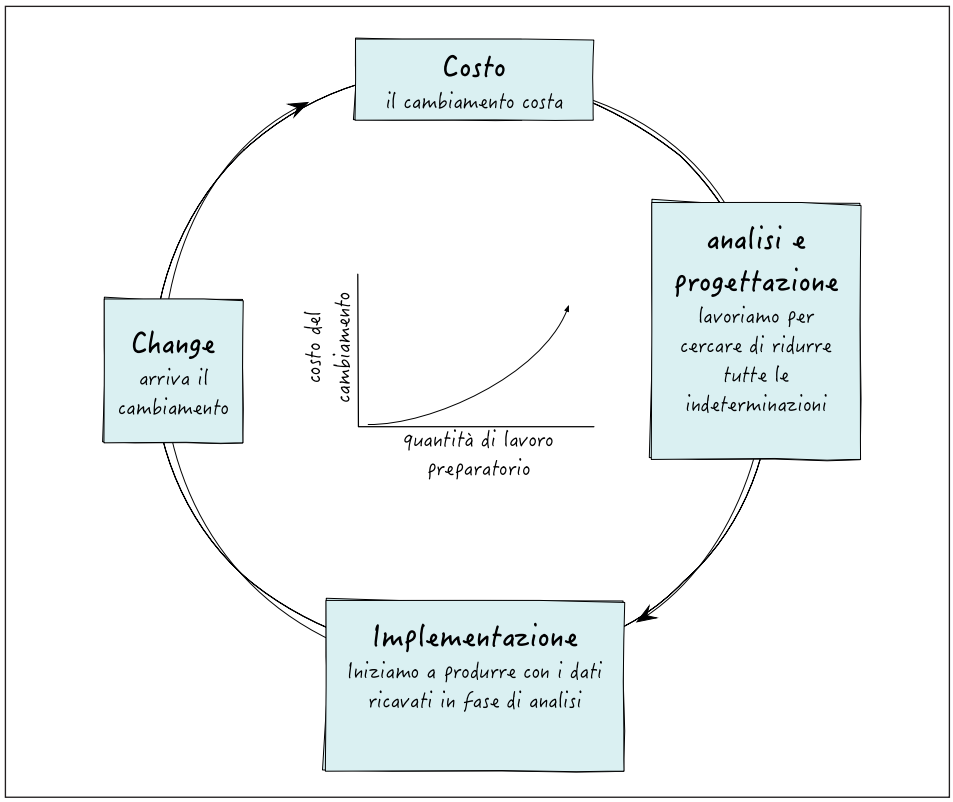


Figura I.7. Il cambiamento costa. In uno scenario complesso, più si cerca di prevedere il futuro lavorando di pianificazione, più il cambiamento avrà un impatto sui costi.

I principi ispiratori del project management classico, prima che il movimento agile iniziasse a metterli in discussione, puntavano a **minimizzare** l'impatto di un eventuale **cambiamento** tramite una stringente pianificazione. Il **paradosso** legato a chi cerca di controllare il cambiamento è ben espresso dallo schema riportato nella figura 7 dove è raffigurato un tipico approccio in cui si cerca di prevenire o minimizzare gli impatti di un cambiamento in un progetto.

In questo caso il cambiamento è inteso come qualcosa di non previsto e per questo viene visto come un costo; per tale motivo ogni sforzo è finalizzato per chiarire ogni dubbio, per far luce su ogni aspetto indeterminato. Questo si traduce spesso, prima di iniziare un progetto, nell'investire copiosamente in **analisi** e **pianificazione**. Purtroppo, poi, in fase di implementazione, molte volte il cambiamento si presenta comunque in modo dirompente, rendendo inutilizzabile buona parte delle ipotesi fatte in precedenza. In questo caso lo spreco diventa doppio: si sono sprecate energie per un lavoro del quale si poteva fare a meno e si è prodotto del materiale inutilizzabile, come i piani di azione. E comunque, sarà necessario cambiare...

Adattarsi ai cambiamenti

Partendo quindi dalla constatazione, corroborata dai molti casi reali, della inutilità di effettuare lunghe sessioni di analisi e pianificazione preventiva, si può provare ad agire con un approccio diverso. Le metodologie agili si pongono esattamente questo obiettivo: provare a ridurre al minimo le fasi di pre-analisi e pianificazione, andando invece direttamente a **produrre qualcosa** che possa dirci se stiamo procedendo nel modo corretto o meno.

Con Scrum per esempio si predilige l'approccio pragmatico: faccio una **piccola parte** di lavoro ed eseguo quanto prima una **verifica** per capire se quanto fatto è quello che il cliente voleva. Ogni errore, o ogni cosa fatta nel modo giusto, serve per correggere la rotta o per confermare che la direzione è quella esatta. Quando arriva un **cambiamento**, non avendo piani da aggiornare e avendo lavorato per piccole iterazioni, lo **spreco sarà minimo**, se non nullo.

Principi dell'Agile Manifesto

Dai quattro valori espressione di tutto il **movimento agile**, il gruppo dei firmatari ha proseguito nella stesura di una serie di **principi** che definiscono più nel dettaglio tali concetti. Vediamoli di seguito.

La nostra massima priorità è soddisfare il cliente rilasciando software di valore, fin da subito e in maniera continua

Dell'importanza del rilascio del **valore** abbiamo già detto. Il "fin da subito" significa cercare fin dalle prime fasi di progetto di rilasciare qualcosa di utile, che soddisfi i bisogni dell'utente, e di utilizzabile dall'utente senza che questo debba aspettare la fine del progetto. In questo modo il team può capire rapidamente se si sta lavorando per colmare i bisogni dell'utente e se lo si sta facendo nel modo richiesto

Accogliamo i cambiamenti nei requisiti, anche a stadi avanzati dello sviluppo. I processi agili sfruttano il cambiamento a favore del vantaggio competitivo del cliente

Pensare di operare in un mondo sempre uguale a se stesso è illusorio. Ogni sforzo per impedire il cambiamento è spreco. Il **cambiamento** è **inevitabile**. Non rispondere in tempi rapidi ed efficaci al cambiamento significa non riuscire a trarre vantaggio da un cambio repentino delle condizioni al contorno. Significa rilasciare un prodotto che probabilmente non soddisfa i bisogni degli utenti perché con molta probabilità questi saranno cambiati. Significa quindi creare un prodotto che non ha valore.

Consegniamo frequentemente software funzionante, con cadenza variabile da un paio di settimane a un paio di mesi, preferendo i periodi brevi

Questo principio spiega come gestire i due punti precedenti: rilasciare **valore** e gestire il **cambiamento**. Sono due obiettivi che si possono ottenere nella pratica grazie a un disciplinato processo di produzione **iterativo e incrementale**. Iterativamente, il team

rilascia qualcosa che risponda alle richieste dell'utente. Incrementalmente, a ogni iterazione il team aggiunge un pezzo al sistema in costruzione. Utilizzare iterazioni brevi consente di avere verifiche frequenti del lavoro che si sta svolgendo: questo permette di individuare prima possibile eventuali cambiamenti dei requisiti o delle condizioni al contorno e di porvi rimedio.

Committenti e sviluppatori devono lavorare insieme quotidianamente per tutta la durata del progetto

Lavorare in **gruppo**, a stretto **contatto**, nello **stesso posto** e con una **bassa rotazione** delle persone sono certamente condizioni che favoriscono il miglioramento delle prestazioni e della qualità del prodotto rilasciato. Riuscire a creare una stretta **sinergia** fra **committente** (ma anche utilizzatori) e **team** che realizza il prodotto facilita molto lo scambio di informazioni e chiarisce velocemente gli obiettivi del lavoro.

Non si tratta di principi "filosofici" lean o di raccomandazioni operative di Scrum o Kanban che ci suggeriscono di lavorare a stretto contatto: è un dato di fatto che sia più efficace, ed è comprovato dall'esperienza quotidiana nello svolgimento di una qualsiasi attività. Gli studi delle scienze sociali e di psicologia comportamentale lo confermano.

Realizzare gruppi di lavoro stabili è uno dei requisiti più difficili da implementare, spesso incompatibile con l'organizzazione dell'azienda e con la politica di crescita professionale dei singoli e dell'organizzazione. È altresì vero che si può fare Agile anche se non si riesce a rispettare in pieno queste raccomandazioni: è probabile che in quel caso non si riuscirà a sfruttare appieno le potenzialità delle persone del gruppo che saranno probabilmente coinvolte a in modo differente.

Fondiamo i progetti su individui motivati. Diamo loro l'ambiente e il supporto di cui hanno bisogno e confidiamo nella loro capacità di portare il lavoro a termine

Se l'imposizione di una rigida disciplina e il rispetto delle regole possono spingere il gruppo a raggiungere determinati obiettivi, persone motivate e fiduciose delle proprie capacità possono fare molto di più. Non c'è regola, o capo che la faccia rispettare, che possa competere con un gruppo di **persone motivate** a collaborare per raggiungere un obiettivo, allineate con la visione di business e del prodotto.

Una conversazione faccia a faccia è il modo più efficiente e più efficace per comunicare con il team e all'interno del team

Il **dialogo** e il confronto faccia a faccia sono probabilmente il modo migliore per trasferire informazioni e quindi per risolvere un problema. In una **conversazione** dal **vivo**, oltre alle parole, si riesce a trasferire una quantità notevole di informazioni legate alla comunicazione paraverbale, visto che molta parte del contenuto informativo non passa solo dalle parole. Mail, chat, telefono e videoconferenze sono strumenti molto potenti e utili, ma non hanno la stessa efficacia di una sana chiacchierata davanti alla macchina del caffè.

Il software funzionante è il principale metro di misura di progresso

Il **software funzionante** è probabilmente la cosa più importante su cui un team agile deve focalizzare la propria attenzione. Cosa c'è di più utile del cominciare a utilizzare il prodotto, man mano che questo prende forma, per misurare i progressi del progetto? Sicuramente non le ore lavorate, le riunioni fatte, i giorni smarcati in un diagramma di Gantt.

I processi agili promuovono uno sviluppo sostenibile. Gli sponsor, gli sviluppatori e gli utenti dovrebbero essere in grado di mantenere indefinitamente un ritmo costante

Nel settore manifatturiero è noto che una macchina non possa essere "spinta" a lungo oltre una soglia dell'80% perché altrimenti si rompe. Gli uomini da questo punto di vista non sono diversi dalle macchine, ma spesso facciamo finta di non accorgercene.

Riuscire a lavorare oltre i limiti fisici del nostro corpo è spesso vista come una capacità di cui vantarsi o che ci fa avere il rispetto da parte dei colleghi. In realtà questo comportamento porta normalmente a un **abbassamento della produttività personale** e quindi a limitare la produttività generale del gruppo. Chiunque abbia svolto per un periodo prolungato un lavoro fatto di ore passate in ufficio oltre l'orario canonico, con un maggior livello di stress, un carico di lavoro oltre la media, sa che poi incorrerà in una ricaduta negativa, in cui la produttività scende al di sotto di tale media, anche se poi si dovrebbe capire cosa si intende per media.

Nel lavoro o nella vita, dosi prolungate di stress riducono sull'immediato le nostre facoltà mentali, mentre sul lungo periodo portano al **burnout**, patologia riconosciuta in ambito medico.

La continua attenzione all'eccellenza tecnica e alla buona progettazione esaltano l'agilità

Chi abbraccia la filosofia agile pone la qualità al centro del proprio lavoro. Qualità significa massimizzare il valore del prodotto che si rilascia al cliente; significa quindi abilitare molti dei punti precedenti. Instaurare all'interno dell'azienda un processo di **continuo miglioramento** del processo di lavoro, di incremento della cultura, informatica ma non solo, di una maggior attenzione agli aspetti relazionali e sociali del gruppo portano a creare prodotti migliori, portano a una maggior attenzione ai bisogni degli utenti finali.

La semplicità è essenziale: l'arte di massimizzare la quantità di lavoro non svolto

Uno degli obiettivi ricorrenti nella filosofia Lean è **ridurre** gli **sprechi**, cosa che si ottiene lavorando su molti aspetti del processo di lavorazione. Un tipo particolarmente frequente di spreco è quello che deriva dalla realizzazione di cose non richieste o comunque non necessarie per rilasciare valore al cliente. Tale spreco ricadrà su chi compra un prodotto a scatola chiusa o su chi sviluppa un prodotto su commissione, ma non

verrà pagato per le cose fatte ma non necessarie. La maggior parte dei prodotti software che usiamo sono farciti di funzionalità che non usiamo ma che abbiamo pagato

Creare **prodotti semplici** è quindi una regola importante in modo da minimizzare il rischio di realizzare qualcosa non necessario. Nel dubbio, si provi sempre a non realizzare una qualche funzionalità; se dovesse servire, si potrà implementarla in un secondo momento.

Le architetture, i requisiti e la progettazione migliori emergono da team che si auto-organizzano

Le metodologie agili non amano fare pianificazione troppo estesa e di lungo termine: questo si ripercuote sia sulla pianificazione delle attività, sia sugli aspetti tecnici. A inizio progetto è difficile sia pianificare date e consegne, che definire nei minimi dettagli ogni aspetto tecnologico e architetturale. Il suggerimento in questo caso quindi è quello di impostare uno **schema architetturale** di **massima**, provvedendo a **completarlo successivamente**, tramite raffinamenti, aggiunte, miglioramenti.

A intervalli regolari il team riflette su come diventare più efficace; tramite l'analisi di quanto fatto, regola e adatta il proprio comportamento di conseguenza

Probabilmente una delle pratiche più importanti delle metodologie agili è la **retrospettiva**. Durante questa attività infatti, non solo si cerca di capire se il prodotto corrisponde a quanto richiesto (valutazione del **cosa** si è fatto), ma si cerca di analizzare e valutare il processo e il modo di lavorare del team (valutazione del **come** lo si è fatto). È nostra intenzione dedicare ampio spazio al tema delle **retrospettive** in una futura edizione del libro riveduta e ampliata.

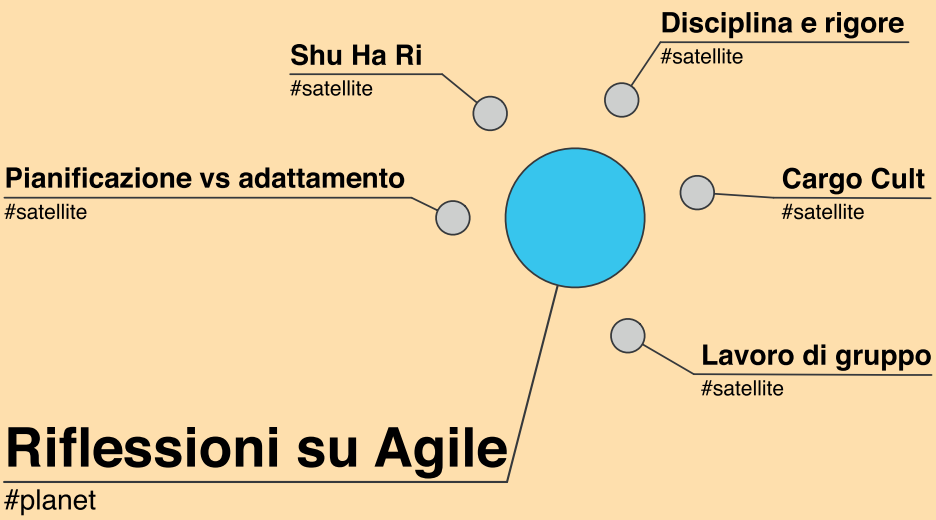
Riferimenti

[FT] F. Taylor, *Principles of Scientific Management*, 1911 (trad. it., *L'organizzazione scientifica del lavoro*, ETAS, 2004)

[KR] K. Rubin, *Essential Scrum: A Practical Guide to the Most Popular Agile Process*, Addison-Wesley, 2012

[MA] Agile Manifesto
http://agilemanifesto.org/

Capitolo 4
Riflessioni su Agile



Introduzione

Concludiamo questa parte di introduzione alla "filosofia" Lean/Agile con una serie di esempi che tentano di spiegare meglio i concetti del movimento agile. Si tratta di riflessioni nate da casi reali, di conoscenze accessorie di supporto e di metafore ormai "classiche", che spesso vengono usate in sede di coaching per chiarire meglio valori e principi espressi dal Manifesto Agile e illustrare certe situazioni in cui si imbatte che affronta l'esplorazione della galassia agile..

Pianificare o adattarsi

Come si è visto nei capitoli precedenti, in un sistema semplice è possibile prevedere la conseguenza di una azione: il risultato è prevedibile e spesso ripetibile. Se un bicchiere di vetro cade per terra da un'altezza di 2 metri, di solito si rompe. È un risultato prevedibile e ripetibile, ovviamente utilizzando un altro bicchiere... Per il dominio complicato le cose sono meno immediate, ma è possibile comunque fare delle previsioni affidandosi alle **buone pratiche**.

Tutt'altra cosa per il dominio **complesso**, dove si può spiegare a posteriori un evento già successo tramite l'interpretazione delle azioni intraprese. In un dominio complesso quindi non solo non è pensabile prevedere nel dettaglio gli effetti delle proprie azioni, ma ci troviamo spesso ad agire su un terreno che cambia frequentemente rispetto a quanto si era immaginato poco prima. Si potrebbe quasi pensare che la strategia migliore per agire sia quella di non avere alcuna strategia predefinita: **adattarsi** allo scenario che ci si prospetta, seguendo il proprio istinto. In realtà, come spesso capita, la soluzione migliore è un mix di questi due estremi (pianificazione preventiva e azione immediata nel corso degli eventi) ed è proprio su questo mix che si basa buona parte della filosofia agile.

Torna utile, in tal senso, una celebre frase del Generale Eisenhower, divenuto negli anni Cinquanta presidente degli USA, la quale recita più o meno in questo modo: "Nel preparare la battaglia, mi sono sempre reso conto che i piani sono inutili, ma la pianificazione è indispensabile". Al di là della frase a effetto, il senso appare abbastanza chiaro, come spiegato bene nel libro *Essential Scrum* [KR], in cui viene riportato l'esempio dello sci estremo.

Sci estremo

L'autore vuol sapere da un suo amico, appassionato di tale disciplina, quale sia il modo migliore per scendere con gli sci da una montagna innevata, in mezzo a valloni, tra rocce e poi boschi. La domanda che viene posta è la seguente: "Come ti prepari per una discesa estrema? Immagino che tu svolga un minuzioso lavoro di preparazione, pianificando ogni passaggio della tua discesa con gli sci; altrimenti come potresti rimanere incolume dopo tutti i rischi che ti prendi?".

La risposta, illuminante nella sua semplicità, fornisce un'indicazione molto precisa su come si dovrebbe **pianificare** il lavoro in un team **agile**: "Pianificazione minuziosa?

No, sarebbe impossibile farlo per preparare una discesa estrema. Troppe variabili e troppi imprevisti. Quando sono in cima alla montagna, cerco di visualizzare il punto di arrivo e di tracciare una rotta indicativa che mi ci porti. Ma oltre a questo, ogni ulteriore pianificazione sarebbe inutile: dalla cima della montagna infatti è impossibile vedere ogni passaggio, scorgere ogni avvallamento del terreno, valutare tutte le possibili alternative. Se anche riuscissi a sorvolare la parete con un elicottero in modo da riuscire a disegnare una mappa tridimensionale, non mi sarebbe molto utile. Dall'alto non si possono avere indicazioni precise sullo stato della neve o sulla compattezza del terreno. Senza contare che in montagna le condizioni meteorologiche possono cambiare molto rapidamente: al momento della discesa un cambio di vento improvviso renderebbe vana ogni rotta tracciata in precedenza. Per questo, cerco di farmi un'idea di massima di quello che potrò incontrare, ma il vero lavoro di preparazione è mettermi nelle condizioni di affrontare quello che incontrerò al momento, in modo che possa affrontarlo e reagire agli imprevisti. Per questo lavoro molto sulla forma fisica, sulla preparazione tecnica e sull'equipaggiamento".

Questo è esattamente lo spirito con il quale si dovrebbe affrontare un progetto secondo la filosofia agile: lavorare molto sulle **abilità operative** dei singoli, sulle **competenze**, sulla **cultura**. Oltre a questo, è utile preparare una **pianificazione di massima** ad alto livello senza scendere troppo nei dettagli: solo quando inizia l'attività sul campo, ossia quando si iniziano a implementare le funzionalità del prodotto, si potranno avere indicazioni precise su come procedere. L'esperienza diretta infatti permetterà di valutare le differenze fra quanto immaginato inizialmente e la realtà delle cose.

Lo Shu Ha Ri nell'Agile

Tempo fa passai un anno della mia vita praticando il paracadutismo sportivo. Feci un corso in inverno per apprendere le tecniche di base e, non appena il tempo fu clemente, andammo in un piccolo aeroporto per prendere il brevetto di **lancio vincolato**.

In televisione si vedono paracadutisti che si lanciano da 5000 metri e precipitano ad alta velocità verso terra prima di aprire il paracadute, dando l'impressione di nuotare nell'aria; questo modo di fare paracadutismo si chiama caduta libera o Accelerated Free Fall (AFF). L'AFF si eseguiva a quei tempi dopo aver fatto un po' di esperienza con il **lancio vincolato**, anche se in tempi più recenti si tende a saltare questa fase intermedia: con il lancio vincolato, il paracadutista esce dall'aereo e si allontana dal velivolo in caduta **non** libera appunto perché c'è la fune di vincolo, un nastro ad alta resistenza collegato da un lato all'involucro del paracadute, dall'altro all'aereo. Quando il paracadutista esce dall'areo, il nastro si tende e provoca l'uscita del paracadute dalla sua sacca, agevolando in questo modo la sua apertura. In questo caso il paracadutista rimane nel vuoto in caduta libera solo per pochi secondi, che la prima volta sembrano un'eternità; quasi subito il paracadute inizia a gonfiarsi "trattenendo" il passeggero in una discesa controllata.

Questo è il tipo di lancio che fanno anche i militari quando vengono paracadutati dietro le linee nemiche: in quel caso non è presente la componente sportiva o ludica,

per cui la parte di caduta libera è assente, a parte casi specialistici in cui l'aereo non può avvicinarsi alla zona designata per l'atterraggio dei paracadutisti, e allora i soldati compiono delle lunghe planate in caduta libera. Nel mio caso di studente squattrinato non troppo impavido, esauriti i soldi e il coraggio mi fermai al primo brevetto, quello con la fune di vincolo.

La metafora del paracadutismo: la chiave è nell'addestramento

Che c'entra tutto questo con l'Agile? Mi è venuto in mente perché, durante quell'inverno passato in palestra, il nostro istruttore ci preparò molto duramente tanto che, quando andammo a fare l'esame, ci fecero i complimenti per il livello di preparazione teorica ma soprattutto pratica.

Cosa avevamo fatto di così speciale da meritarci i complimenti degli esaminatori? Col senno di poi direi che non avevamo fatto nulla particolare, se non leggere attentamente un manuale che conteneva i principi base di meteorologia, aerodinamica e ovviamente paracadutismo. Il tutto accompagnato da una lunghissima ed estenuante preparazione pratica; probabilmente fu questa attività di pratica ripetuta mille volte che fece la differenza.

Ricordo le ore passate in palestra a provare centinaia di volte la manovra di uscita dall'aereo, la manovra di emergenza, le tecniche di caduta e altro ancora, in modo che diventassero per noi gesti automatici, quasi istintivi. Per la manovra di uscita dall'aereo avevamo passato ore e ore a lanciarci da una piccola struttura in metallo su un materasso, passando dalla posizione seduta — si sta sul portellone dell'aereo con le gambe penzoloni nel vuoto — a una posizione a volo d'angelo, in pratica una specie di X fatta con braccia e gambe aperte in maniera adeguata.

La posizione doveva essere quanto più simmetrica per mantenere l'assetto stabile ed evitare così di iniziare a roteare nel vuoto mentre il paracadute si apre: non è bello fare la trottola proprio mentre 100 m^2 di tela si aprono e 100 m di cordino si srotolano intorno a te. In figura 8 è illustrata tale procedura: si noti la posizione a "volo d'angelo" e la fune di vincolo attaccata al velivolo, che aprirà il paracadute.

L'altra cosa che ci fecero provare per ore fu la manovra di sgancio del paracadute principale, manovra che deve essere effettuata tutte le volte che il paracadute principale non si apre bene, o che si rompe qualcosa. In tal caso il paracadutista tira una leva di emergenza che si trova da un lato dell'imbracatura e che provoca lo sgancio del paracadute. In quel momento riprende la caduta libera; passato qualche secondo necessario perché il paracadute sganciato si sia allontanato nel vuoto, si tira la seconda leva che porta all'apertura del secondo paracadute di emergenza. È superfluo ricordare cosa possa succedere se si sbaglia la sequenza... aprendo il secondario prima di aver sganciato il principale.

Questi movimenti devono diventare meccanici e istintivi perché nel momento della crisi è difficile restare lucidi per eseguire il ragionamento con cui effettuare la sequenza di emergenza. Nel mio caso non ho mai avuto problemi di questo tipo, anche se mi

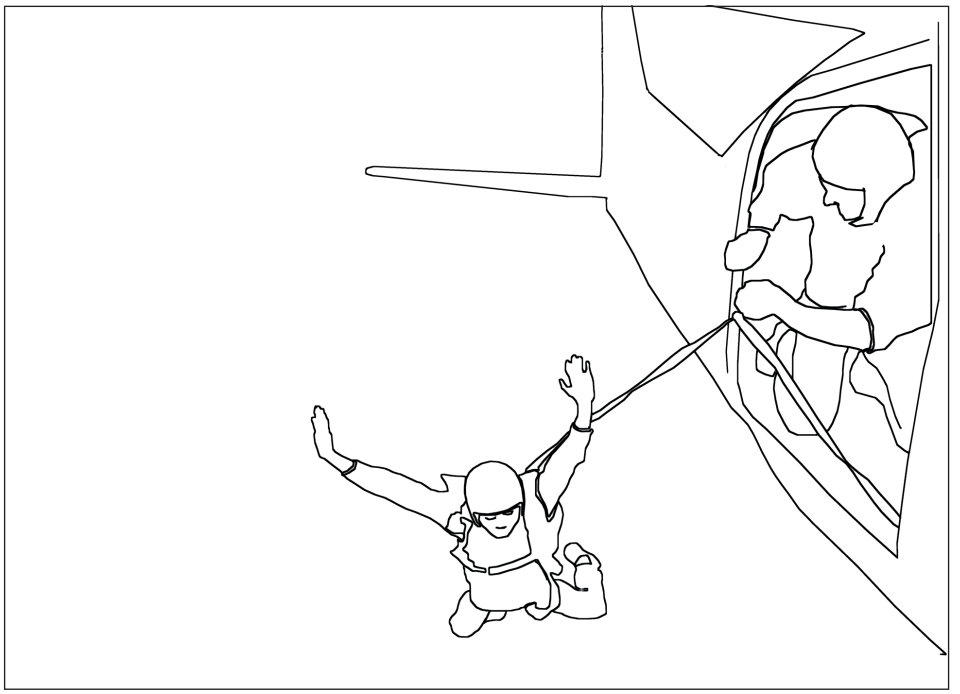


Figura I.8. L'uscita dall'aereo per il lancio con il paracadute. Si noti la posizione a "X" che garantisce la stabilità, e la fune di vincolo che apre il paracadute.

dissero che al primo lancio non avrei capito nulla e che a malapena avrei visto l'aereo che si allontanava. Confermo, non mi ricordo nulla.

Interessante anche un altro aspetto, quello del ripiegamento del paracadute: ogni paracadutista ripiega il proprio paracadute dopo il lancio; per i lanci successivi però i paracadute sono prelevati a caso da un magazzino, in modo che non si sappia chi ha piegato il tuo. Massima fiducia e rispetto deve esserci infatti fra i paracadutisti di una stessa compagnia o gruppo. Questo ci spingeva a porre la massima attenzione nella procedura di ripiegamento; paradossalmente mi sarei fidato meno di un paracadute ripiegato da me. A pensarci dopo molti anni, questo è un concetto molto agile: il *trust* nel gruppo di lavoro.

Il percorso di apprendimento: Shu Ha Ri

Ho citato la storia del brevetto di paracadutismo non tanto per dare un'aura di "impavido spericolato" all'agilista, ma perché si aggancia in modo piuttosto diretto con alcuni principi dell'agile. Uno degli insegnamenti dell'agile infatti è quello di seguire un **percorso di apprendimento** che parte dall'acquisizione delle pratiche in modo "meccanico", per certi versi un po' scolastico, per arrivare successivamente alla completa comprensione del significato dei vari gesti.

Solo quando si possiede la completa **padronanza** delle tecniche e della loro applicazione, infatti, si può iniziare un processo di approfondimento dei principi teorici che stanno dietro ciò che fino a quel momento si era appreso in modo piuttosto meccanico. Questa fase è quella che dà piena consapevolezza dei principi e del senso di quello che si sta facendo.

A questo punto si approda alla parte più profonda e importante, quella in cui si prova a vedere cosa succede se si aggiunge un pezzetto al metodo, se si prova a personalizzare la pratica. Questa fase può avvenire solo al termine di un percorso in cui si deve prima di tutto comprendere l'importanza della tecnica sviluppata e ottimizzata da altri prima di noi. In una frase, essere agili significa anche realizzare quello che in giapponese si chiama **Shu Ha Ri**.

Lo **Shu Ha Ri** è un percorso di crescita, diviso in tre fasi e deriva dalle arti marziali giapponesi, in particolare dall'Aikido; il concetto di Shu Ha Ri è stato portato nel mondo agile grazie al celebre Alistair Cockburn. Martin Fowler, altro nome noto ai più, in un suo blog ce la racconta in questo modo (nostra traduzione):

- **Shu**: in questa fase iniziale, lo studente segue scrupolosamente gli insegnamenti di un maestro. Si concentra sul modo in cui compiere l'azione, senza preoccuparsi troppo dei principi teorici che stanno alla base. Sebbene possano esistere diverse varianti sul modo in cui compiere l'azione, egli si concentra solo sul modo che gli viene insegnato dal suo maestro.
- **Ha**: a questo punto lo studente comincia ad ampliare i suoi orizzonti. Una volta che la pratica base funziona, comincia a imparare i principi e la teoria che ne rappresentano i fondamenti. Inoltre, comincia ad apprendere anche da maestri diversi e a integrare tali insegnamenti nella sua pratica
- **Ri**: infine lo studente non apprende più dagli insegnamenti ricevuti da altre persone, ma dalla sua stessa pratica. Crea il suo personale modo di affrontare le situazioni e adatta ciò che ha appreso alle particolari circostanze in cui si trova.

Una descrizione più approfondita del processo Shu Ha Ri si trova nel libro *Agile Software Development* [ASD].

Disciplina e rigore

A volte si sente dire quindi che solo tramite una **ferrea disciplina** si può essere agili, ossia solo tramite una rigorosa aderenza ai precetti e alle raccomandazioni della metodologia. Chi vuole lavorare con queste metodologie deve rispettare certe regole, svolgere determinate "cerimonie", attivare specifici ruoli e seguire alcune abitudini.

In questo senso una delle cose che si criticano a una metodologia come Scrum è quella di essere troppo regolata e quindi difficile da adottare in organizzazioni già esistenti. Perché dobbiamo fare tutte le mattine il *daily scrum meeting*? Perché sempre in piedi? Perché la demo si fa sempre il venerdì mattina? Perché la retrospettiva il venerdì pomeriggio? Perché adesso abbiamo uno *Scrum Master*? Perché dobbiamo invitare gli utenti alla demo?

Potrebbe sembrare quindi che Scrum, che è la metodologia agile più diffusa e che dovrebbe fare dell'adattabilità e della flessibilità il suo cavallo di battaglia, sia in realtà estremamente **rigida** e predefinita.

In realtà Scrum ha poche regole, se paragonate ad altre metodologie non agili come RUP; e queste poche ma chiare regole sono state definite proprio per consentire di adattare il lavoro al contesto di progetto in cui si opera, di adeguarsi ai cambiamenti che si presentano, di massimizzare le prestazioni del gruppo in uno scenario incerto e in continua evoluzione.

Il consiglio che si dà quindi, quando un team o una organizzazione deve procedere nell'adozione di una metodologia agile, è di seguire le indicazioni e le regole proposte nel modo più rigoroso e fedele possibile, anche se questo può apparire un controsenso.

Col tempo si potrà valutare se sia il caso di rilassare qualche vincolo — per esempio provare a fare il daily scrum meeting in video conferenza per consentire a colleghi di partecipare da remoto — consci del fatto che ogni deroga equivale a un piccolo abbassamento delle prestazioni del gruppo di lavoro, a una minor capacità di gestire gli imprevisti o a una minore possibilità di godere dei benefici dell'agilità.

Per rispondere quindi alla domanda iniziale, si può certamente dire che le metodologie agili richiedono il rispetto disciplinato di **determinate regole**. Essere agili vuol dire adottare rigorosamente un modo di essere e di pensare al fine di garantire la massima flessibilità e adattabilità nell'operatività.

Comportamenti privi di consapevolezza: cargo cults

Un importante aspetto da tenere in considerazione quando si decide di adottare una metodologia agile all'interno della propria azienda o gruppo di lavoro è quello di esercitare le giuste pratiche senza dimenticare i principi e i valori che ci stanno dietro: riprendendo quanto detto nel paragrafo precedente, non è bene fermarsi allo Shu dello Shu Ha Ri.

È infatti importante adottare le pratiche e le "cerimonie" delle metodologie agili, ma questo va fatto anche approfondendo i fondamenti teorici, i valori e i principi che ne sono alla base, e progredendo ulteriormente verso una comprensione sempre più profondo di questa "filosofia" operativa. Infatti, il rischio che si corre è quello di **scimmiottare** le cerimonie e le pratiche senza comprenderne completamente il significato: nel gergo dei praticanti delle metodologie agili, si usa il termine di *agile monkeys* per riferirsi scherzosamente a queste situazioni.

Ma, a spiegare ulteriormente questo fenomeno, ben si presta un fenomeno osservato nello scorso secolo, a partire dalla Seconda Guerra Mondiale, e studiato approfonditamente dall'antropologia culturale: si tratta dei cosiddetti **cargo cults** o "culti del cargo".

La trappola della "cargo cult science"

Nel 1974, il fisico statunitense Richard Feynman, in un discorso al California Institute of Technology, mise in guardia i ricercatori dal non diventare "scienziati da culto

del cargo" [FEY]. Feynmann coniò questa espressione rifacendosi a un tema ben noto agli antropologi culturali, ossia quello dei cosiddetti *cargo cults*, un fenomeno religioso affermatosi in alcune società tribali dopo i ripetuti contatti con società tecnologicamente più sviluppate.

I casi meglio documentati riguardano l'area del Sud Pacifico, in particolare nelle isole della Nuova Guinea e nell'arcipelago delle Nuove Ebridi dove, soprattutto nel periodo precedente la Seconda Guerra Mondiale e poi durante tutto il conflitto, arrivarono enormi quantità di beni moderni, paracadutati o fatti atterrare con **aerei cargo** e destinati alle forze britanniche, australiane, neozelandesi e soprattutto americane di stanza in quei territori. Questi prodotti moderni si diffusero anche fra le popolazioni native, determinando un drastico cambiamento nella cultura materiale e negli stili di vita degli indigeni [WOR].

Con la fine delle ostilità, le basi aeree furono abbandonate e di conseguenza si interruppe il rifornimento di merci via cargo. **Non comprendendo le ragioni geopolitiche** della fine di tali rifornimenti, i nativi svilupparono una serie di cerimonie, basate su **riti magici di "imitazione"** delle pratiche e dei comportamenti che gli indigeni avevano

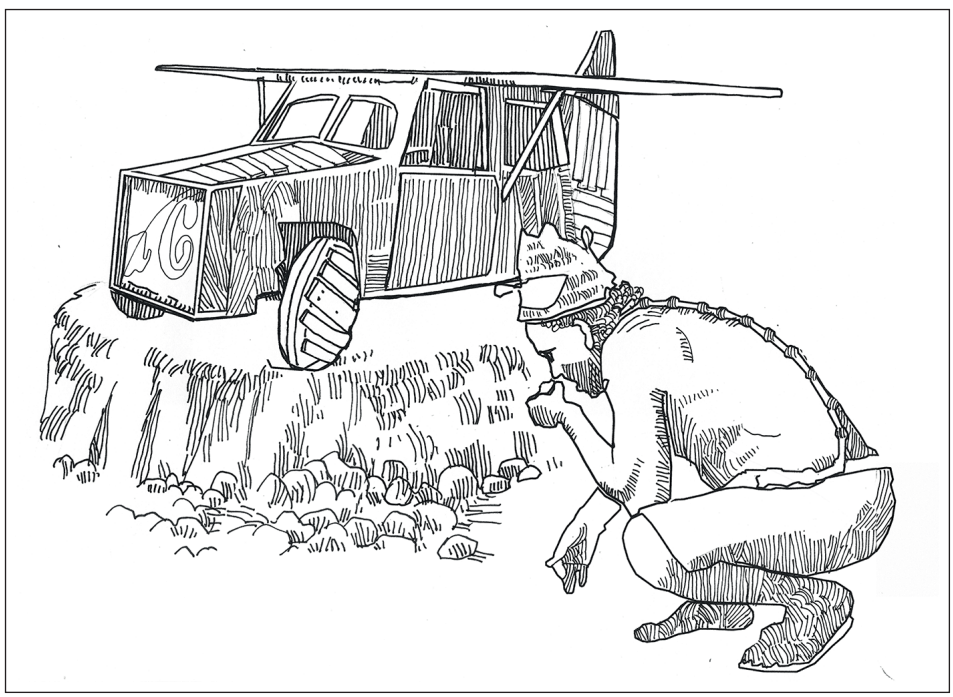


Figura I.9. I nativi adepti dei vari "culti del cargo" misero in atto delle cerimonie di "imitazione" dei comportamenti che avevano osservato presso i militari alleati. In certi casi si arrivò a realizzare dei veri e propri simulacri di aerei, costruiti con i materiali a disposizione, come legno e bambù.

visto compiere da parte dei militari alleati: sono documentati casi in cui furono costruiti simulacri di apparecchi radio con cuffie e microfoni, che poi erano utilizzati dentro finte torri di controllo costruite in legno e bambù. Furono riprodotti degli "aeroplani" in materiali di fortuna, furono create delle rudimentali piste di atterraggio illuminate con torce e altri segnali. Tali culti ebbero forte diffusione nel decennio immediatamente successivo alla fine della Seconda Guerra Mondiale, per poi declinare gradualmente negli anni successivi [CAR]. In ogni caso, dopo molti anni, sopravvivono ancora in alcune aree della Melanesia celebrazioni che mettono in scena parate militari alle quali gli indigeni avevano assistito nel periodo dell'occupazione.

Alcuni di questi culti si sono evoluti e differenziati, con sfumature varie, in veri e propri movimenti millenaristi, tra i quali spicca il cosiddetto "culto di John Frum" il cui nome pare derivare dalla corruzione della frase "I am **John from** America"; gli adepti di tali movimenti, ancora in anni recenti, continuano a svolgere le loro cerimonie, tra le quali delle imitazioni di parate militari.

La lezione di Feynmann

Con "scienziati da culti dei cargo" quindi, Feynmann intendeva indicare quegli studiosi che imitano comportamenti scientifici, restando però in realtà solo alla superficie, **senza comprendere le motivazioni profonde** del metodo e degli strumenti che adottano. Per non essere *cargo cult scientists*, occorre stare sempre in guardia nei confronti delle suggestioni e delle illusioni, mettere sempre in dubbio e sotto attento scrutinio i risultati delle proprie ricerche, e valutare i possibili punti deboli delle varie teorie, evitando di adottare in maniera acritica i risultati delle ricerche di altri scienziati, se questi non passino una attenta valutazione.

A partire dall'esortazione di Feynman, più genericamente oggi il termine **cargo cult** è utilizzato quando si applichino superficialmente a una disciplina le tecniche e le pratiche di un'altra materia, aspettandosi una sorta di misteriosa magia che porti inspiegabili benefici, senza rendersi conto che è necessario impiegare determinati strumenti solo se si ha una profonda **consapevolezza** delle loro caratteristiche e dei loro princìpi.

Il culto del cargo e l'adozione delle metodologie agili

Tornando alle metodologie agili di sviluppo del software, la tematica del culto del cargo viene spesso tirata in ballo quando si parla dell'**adozione** delle metodologie agili in una organizzazione. Fare Scrum **meccanicamente** o disegnare una kanban board **senza** aver chiari i **valori** e i **principi** profondi che vi stanno dietro è un po' come costruire una "radio" di legno e gusci di cocco su una spiaggia del Sud Pacifico, aspettandosi che arrivi un aereo cargo pieno di cibo e altri prodotti. È quello che fecero i manager della Ford quando copiarono il Toyota Production System senza avere un'appropriata conoscenza del **lean**, dando luogo al cosiddetto **Lean di Chicago**.

Fare **cargo cult** in Scrum significa, per esempio, organizzare gli sprint, fare planning e tutte le cerimonie senza comprenderne realmente i valori fondanti, cercando poi spesso

una scusa per derogare a queste pratiche. Questo può significare interpretare in modo **superficiale** le cerimonie, i ruoli e le altre **pratiche**.

Spesso si sente dire: "In azienda siamo agili ormai da un anno; bella roba, anche se, per come viene 'venduto', devo dire che non ci vedo tutta questa rivoluzione e questi benefici". Queste affermazioni dimostrano un approccio limitato in cui è probabile che non si riesca a ottenere il famoso incremento delle prestazioni pubblicizzato da Scrum.

Il lavoro di gruppo

Un aspetto spesso molto discusso, quando si parla di team agile, è quello legato al coinvolgimento del **gruppo** sul progetto. Ci si chiede se sia necessario impiegare persone a tempo pieno a fare sempre la stessa cosa e come conciliare questa cosa con il quotidiano lavoro in azienda, composto spesso di molte attività frazionate e slegate dalla lavorazione su progetto: da rispondere alle mail, a partecipare a incontri per un altro progetto, all'andare dal cliente per fare assistenza o pianificare un nuovo lavoro.

Per rispondere a questi interrogativi conviene forse partire da un esempio preso in prestito dallo sport, e che funziona abbastanza bene per comprendere il problema dai vari punti di vista. L'esempio è quello della staffetta 4 × 400, ma potrebbero andare bene altri sport in cui la ricerca della prestazione si basa sia su allenamento fisico individuale ma anche sul **lavoro di squadra**, **coordinamento**, **sincronizzazione**.

Cosa serve a una squadra per vincere una gara di staffetta? Moltissime cose e, se si esclude il doping, molte sono inerenti al lavoro di team: serve sì allenamento fisico, ma anche una perfetta conoscenza dei meccanismi di base, messa in pratica con un estenuante lavoro sul campo; serve un perfetto affiatamento personale, tanto che spesso gli atleti devono essere anche ottimi amici. Un team quindi, per essere performante, dovrebbe sempre lavorare in determinate condizioni che rappresentano l'ottimale.

Quindi stiamo dicendo che un team Scrum, per lavorare al massimo delle proprie possibilità, dovrebbe sempre poter lavorare in un **ambiente stabile**, senza turnover delle persone, il più protetto da fattori esterni di disturbo.

È anche vero però che queste condizioni non sempre si possono verificare. Spesso le organizzazioni hanno esigenze di più ampio respiro, rispetto alla performance del team. Un'azienda, per esempio, potrebbe porsi come obiettivo quello di saper rispondere alle richieste dei clienti oppure di spostare le persone su progetti diversi per permettere loro di imparare quante più cose possibile. A livello di organizzazione, questi obiettivi si possono ottenere grazie a una sapiente e controllata rotazione delle persone all'interno dei vari gruppi.

Se per esempio non si spostasse Tizio, esperto di una qualche tecnologia o tematica, dal team A al team B, per l'azienda potrebbe essere difficile lavorare alla realizzazione di un nuovo prodotto. Quindi dire: "Il turnover è sbagliato perché abbassa le prestazioni del team" certamente è corretto se si prendono in considerazione solo le esigenze del team e del progetto, ma perde di significato se si guardano le cose da un punto di vista più ampio, quello dell'organizzazione.

In conclusione

Abbiamo voluto riportare in questo capitolo una serie di riflessioni sparse e "laterali" che aggiungono sfaccettature e significati al panorama dei princìpi e dei valori che animano la filosofia lean e le metodologie agili. Lo spirito è stato quello di riflettere ulteriormente sui numerosi fattori che interessano questo approccio, ribadendo da un lato la sensatezza e la validità di intraprendere una transizione verso Agile, ma mettendo d'altro canto in guardia nei confronti di una sbrigativa banalizzazione nell'avvicinarsi a queste discipline.

Riferimenti

[ASD] A. Cockburn, *Agile Software Development*, Addison-Wesley Professional, 2001

[FEY] R. Feynman, R. Leighton (eds), *Surely You're Joking, Mr. Feynman!: Adventures of a Curious Character*, W.W. Norton & Company, New York, 1997

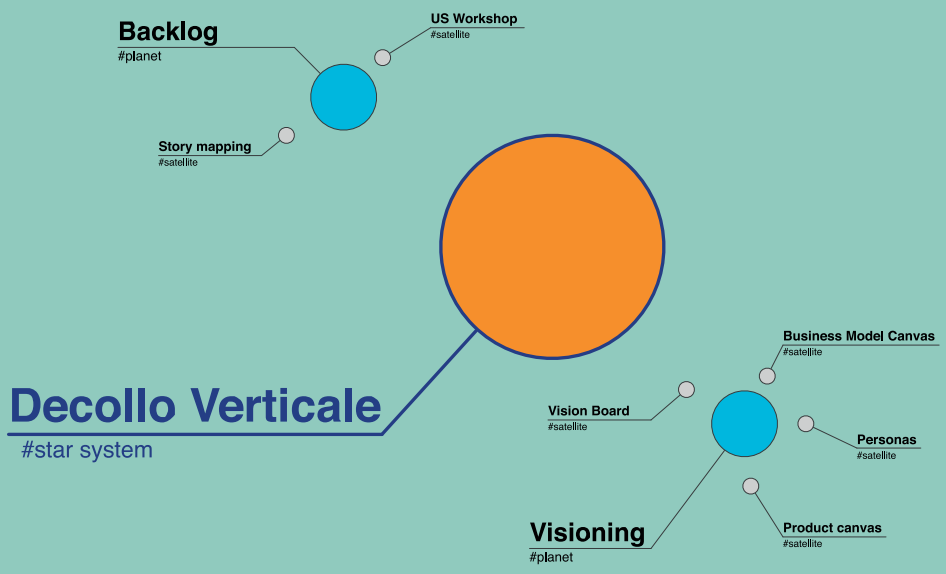
[CAR] L. Lindstrom, *Cargo Cult: Strange stories of desire from Melanesia and beyond*, University of Hawaii Press, Honolulu, 1993

[WOR] P. Worsley, *The trumpet shall sound: a study of "cargo" cults in Melanesia*, Schocken Books, New York, 1968

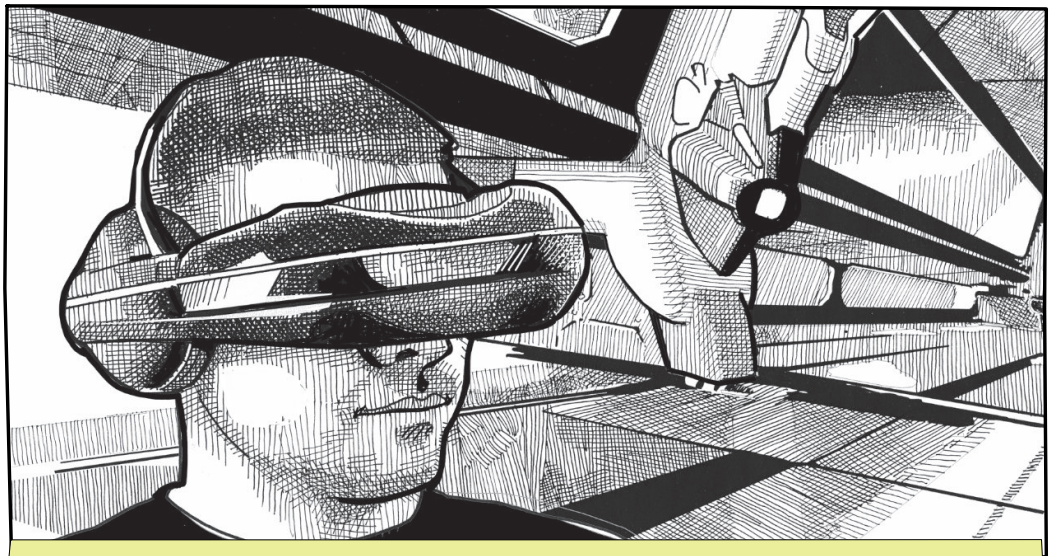
[KR] K.S. Rubin, *Essential Scrum: a practical guide to the most popular agile process*, Addison-Wesley Professional, 2012

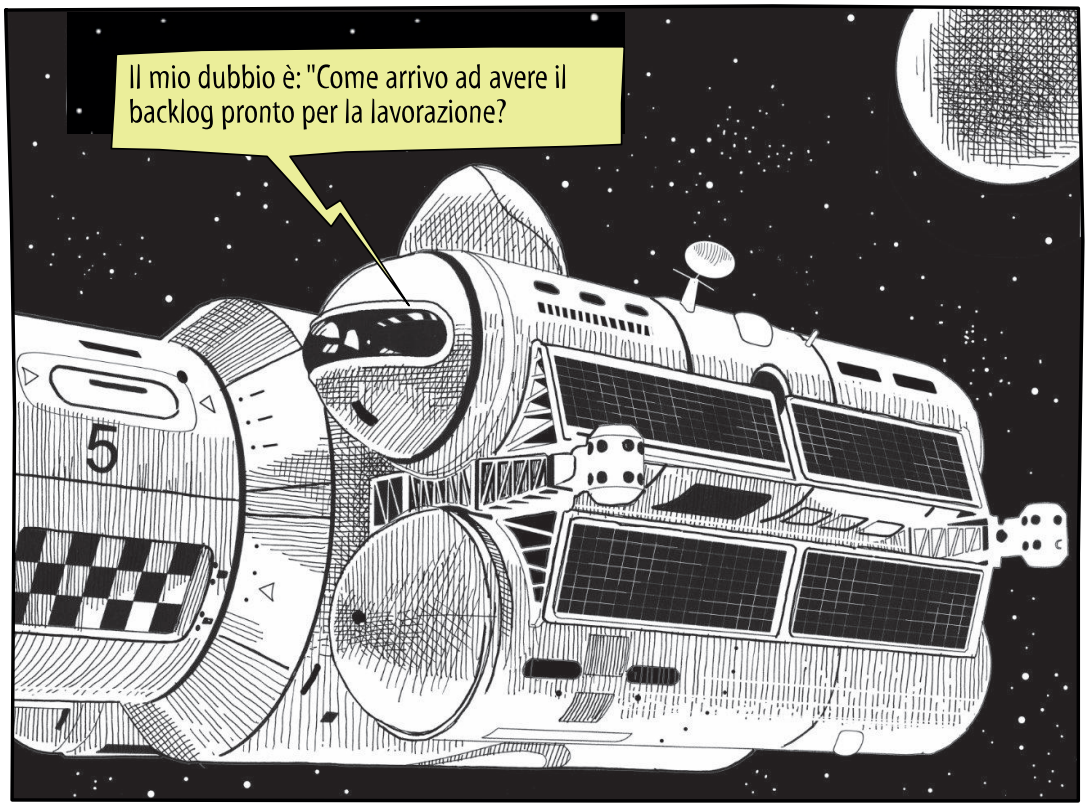
PARTE II
DALLA VISIONE AL PRODOTTO

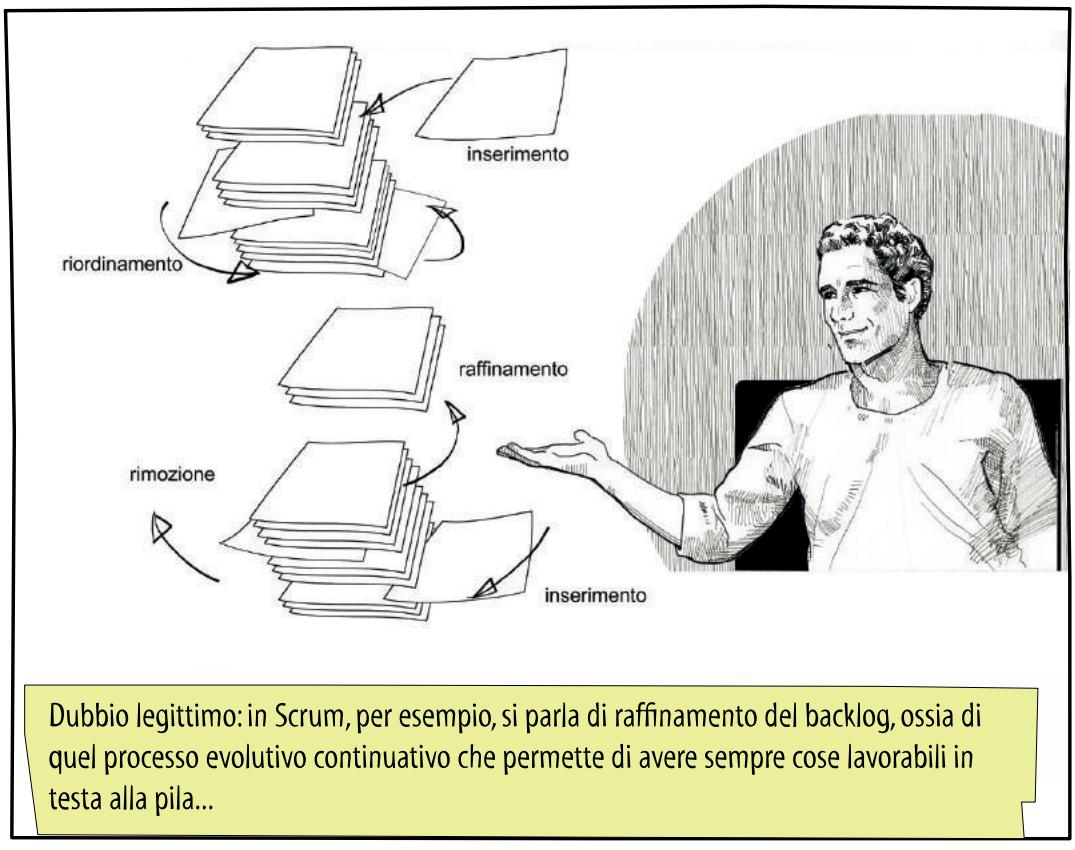
Stefano Leli - Giulio Roggero - Giovanni Puliti



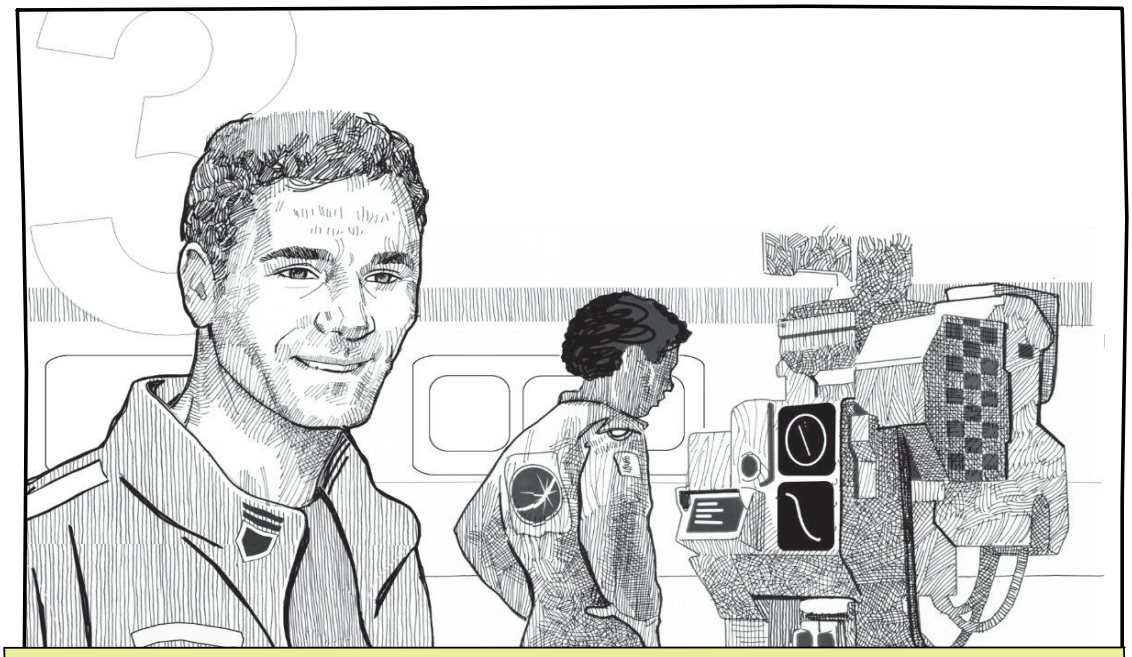
Parte 2
Dalla visione al prodotto



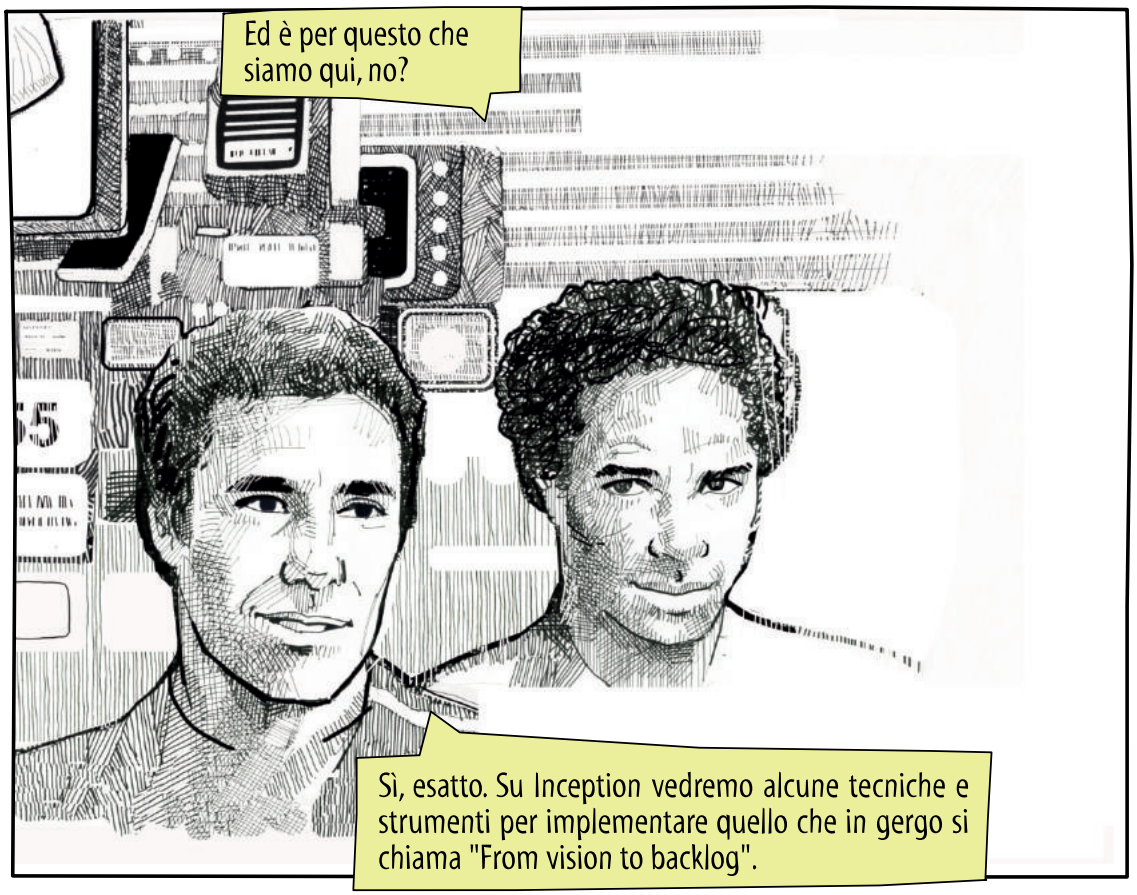




Dubbio legittimo: in Scrum, per esempio, si parla di raffinamento del backlog, ossia di quel processo evolutivo continuativo che permette di avere sempre cose lavorabili in testa alla pila...

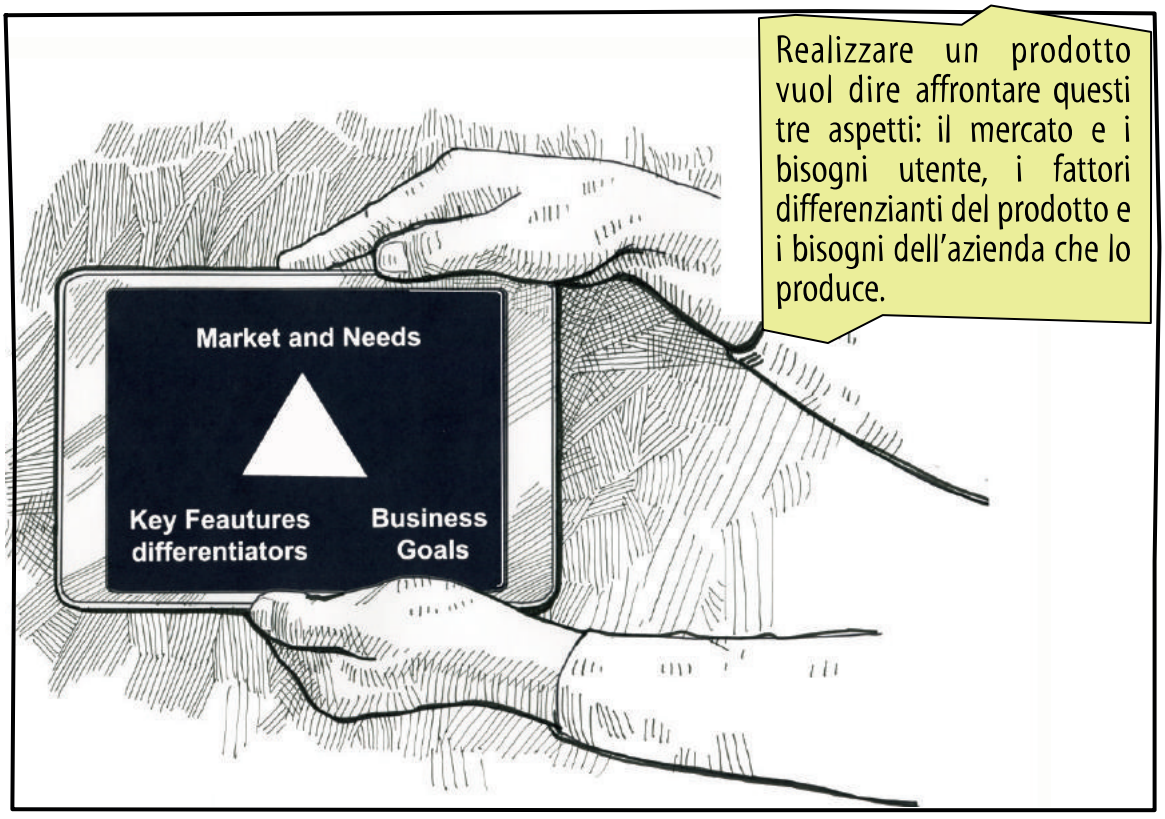


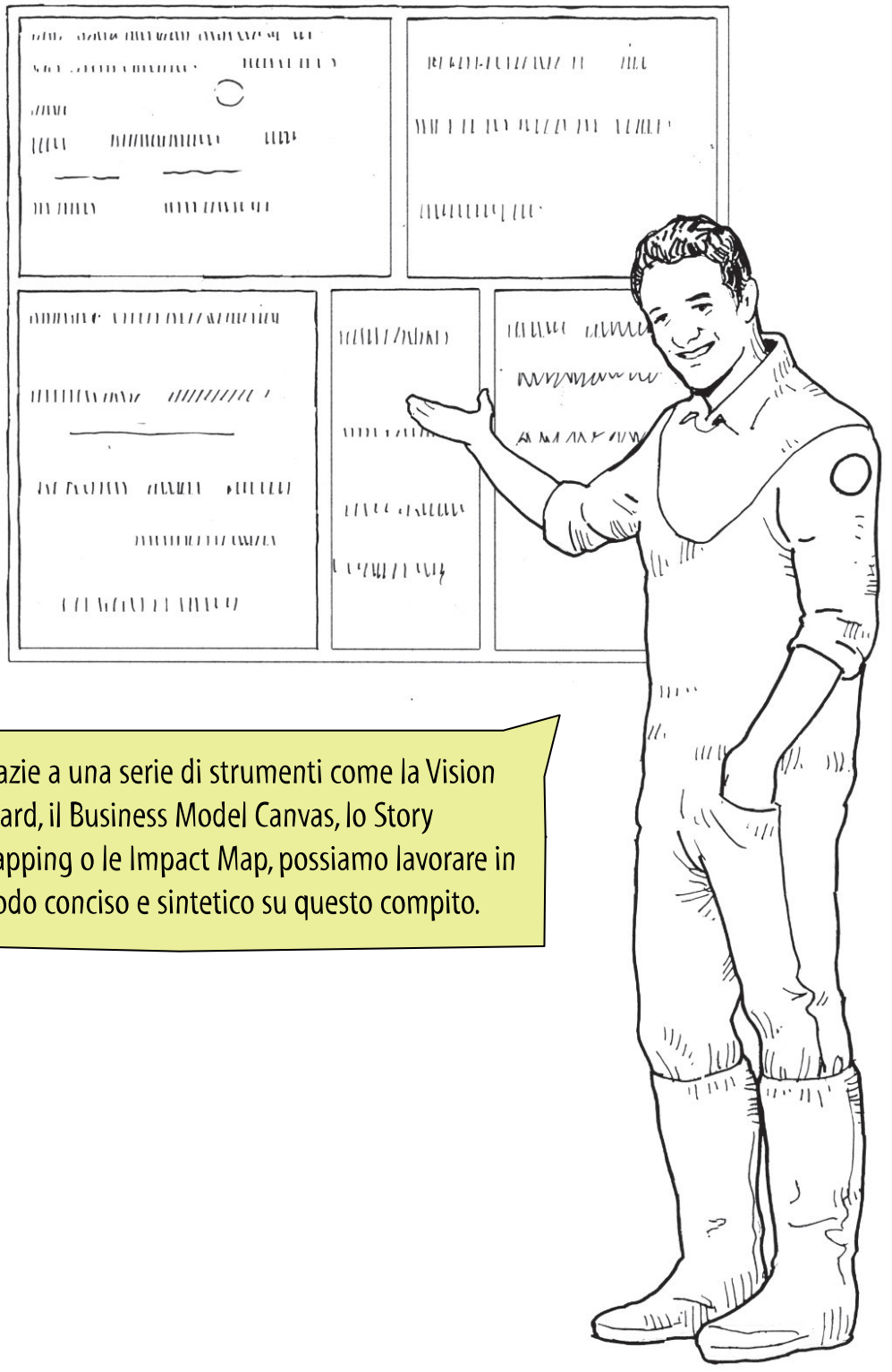
Scrum, ma anche altre metodologie, come Kanban, non si preoccupano della fase iniziale di popolamento e di alimentazione del backlog. Il modo in cui arrivare alla prima versione di questa raccolta delle cose da fare messe in "pila" esula dagli scopi di Scrum…



Questo flusso può essere formalizzato in modo efficace secondo questo schema.

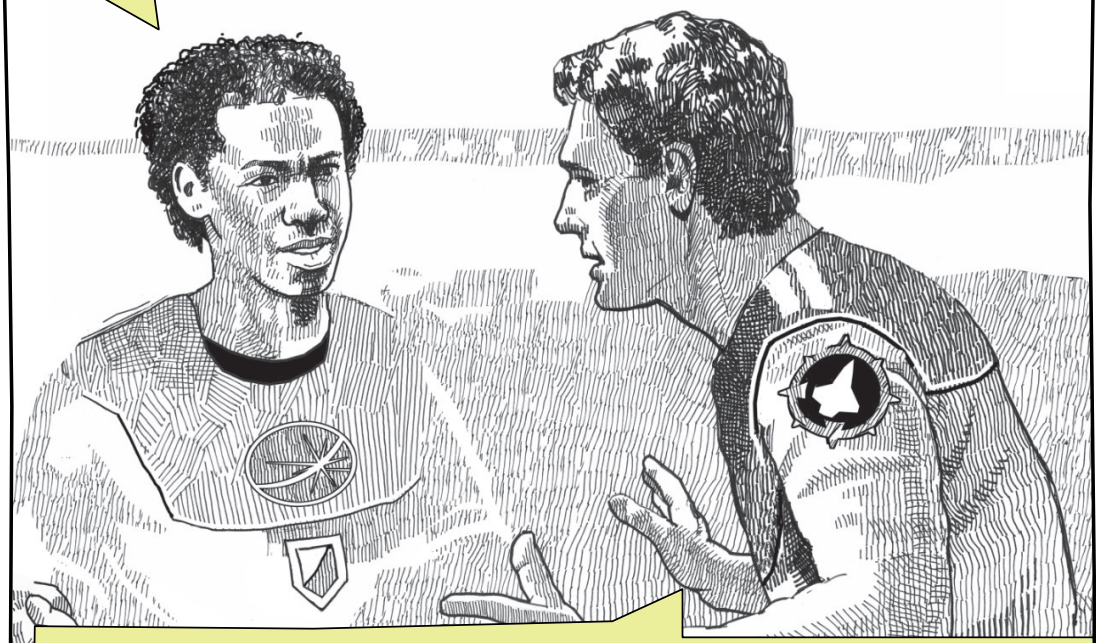
Realizzare un prodotto vuol dire affrontare questi tre aspetti: il mercato e i bisogni utente, i fattori differenzianti del prodotto e i bisogni dell'azienda che lo produce.





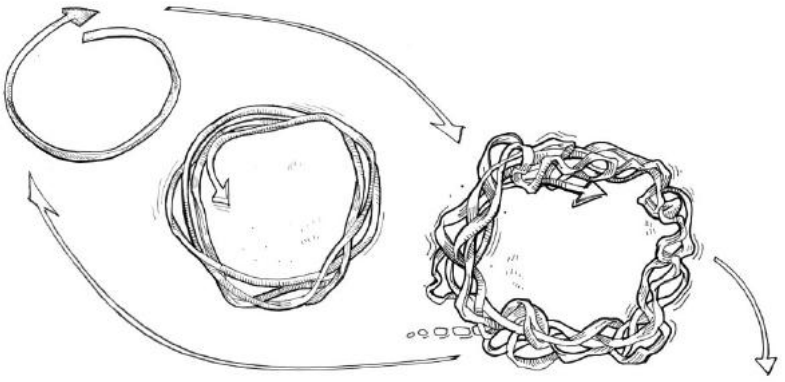
Grazie a una serie di strumenti come la Vision Board, il Business Model Canvas, lo Story Mapping o le Impact Map, possiamo lavorare in modo conciso e sintetico su questo compito.

È vero che questo processo è molto più rapido rispetto al processo tradizionale?

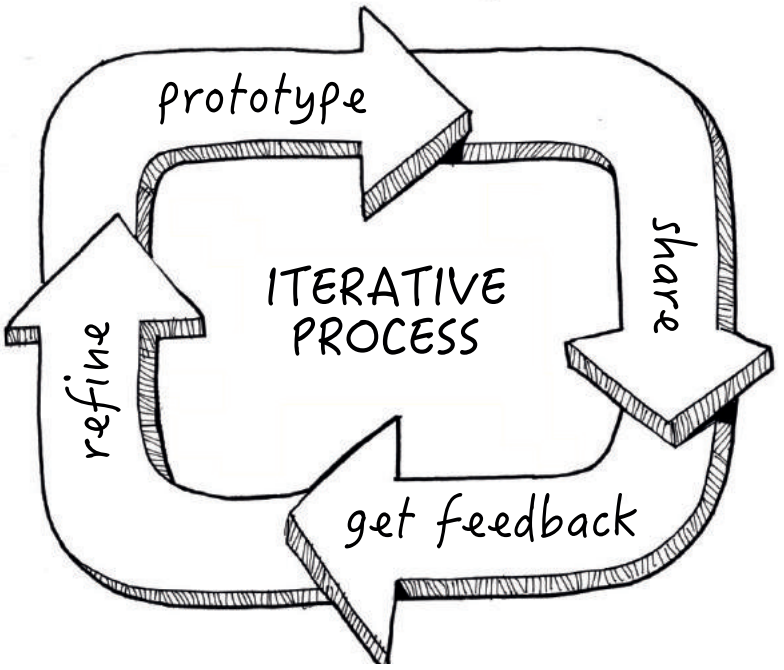


Sì; spesso è una fase rapida, ma non è questo l'aspetto più importante di questo approccio. La cosa più interessante, infatti, è che è un processo iterativo e incrementale finalizzato a produrre in modo rapido qualcosa che possa essere mandato in lavorazione al team di sviluppo. Questo approccio permette di iniziare a sviluppare il codice molto prima rispetto all'approccio classico waterfall.

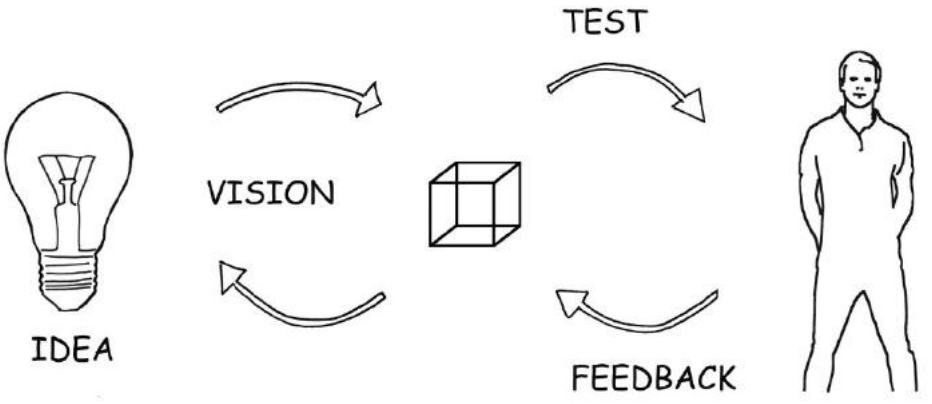
Un po' come se avessimo una Iterazione sulla fase di progettazione delle iterazioni. Più che, "from vision to backlog", sarebbe più corretto chiamarlo "from vision to backlog AND back", a sottolineare che non si tratta di un processo di lavoro monodirezionale a cascata. Anche in questo caso si itera ed si raffina il modo in cui si trasforma l'idea in un prodotto.



Le prime funzionalità che si realizzano sono molto utili per raffinare l'idea di quello che dovrà essere realizzato in seguito.

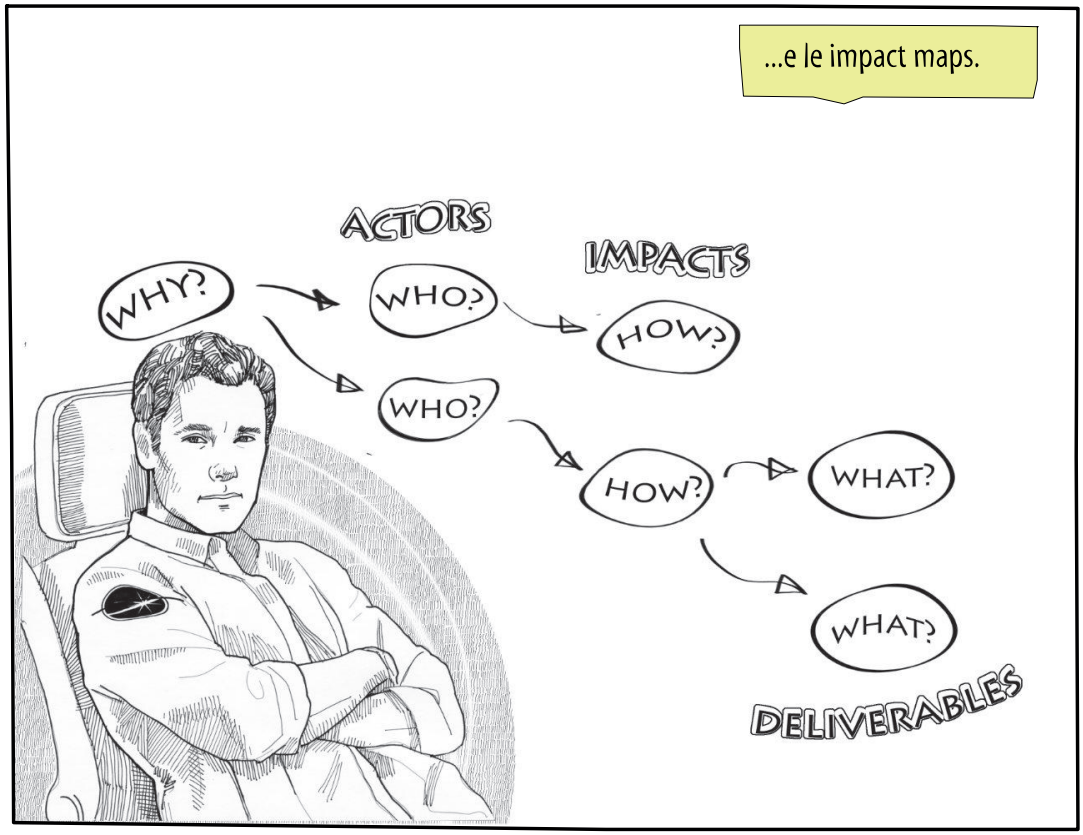


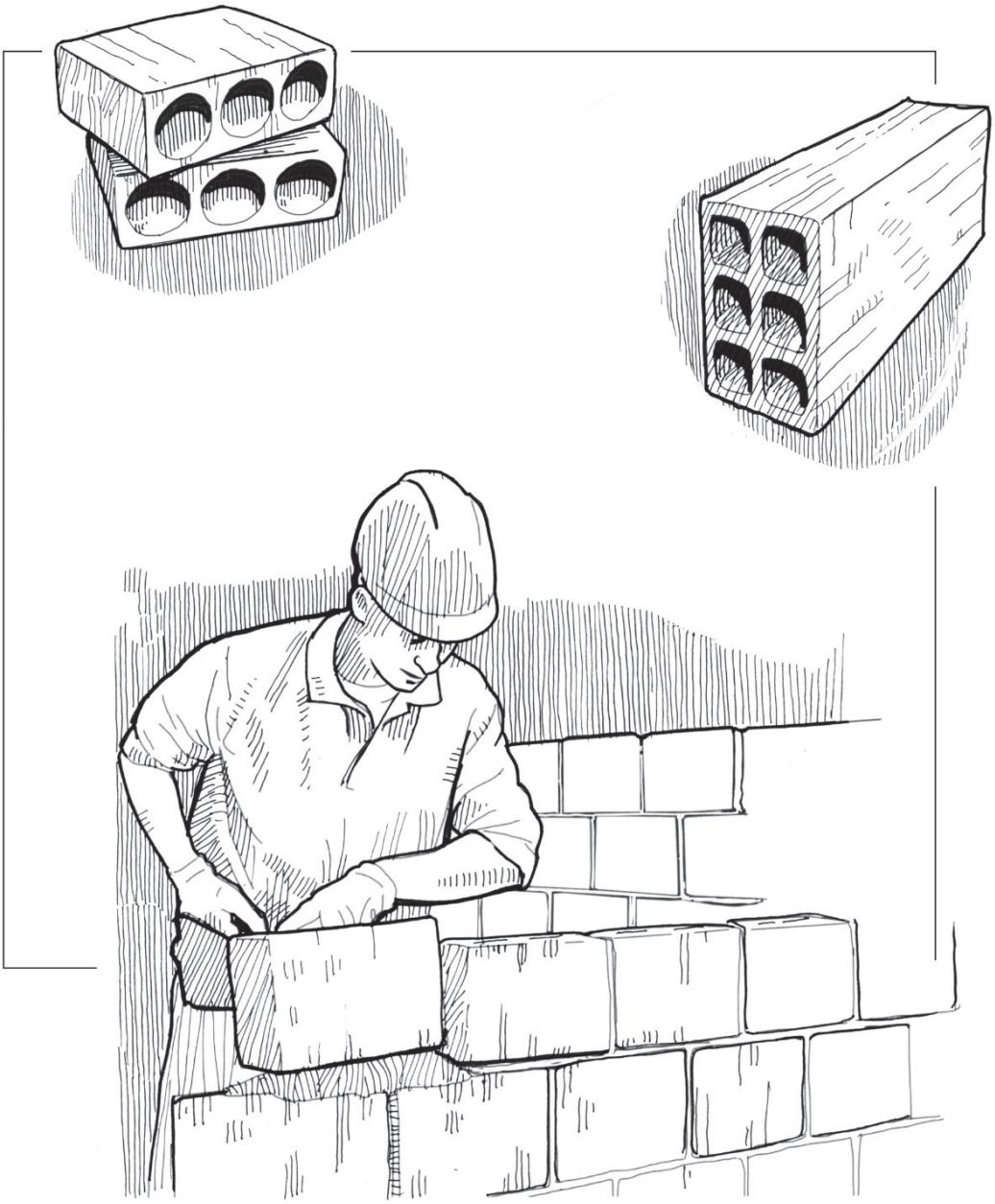
I feedback che si otterranno dai primi rilasci permetteranno di affinare le fasi successive di trasformazione di idee in funzionalità da implementare.



Infatti, se è pur vero che la vision di prodotto non dovrebbe cambiare più di tanto, i test con gli utenti finali permetteranno di cambiare opinione sui dettagli del prodotto.

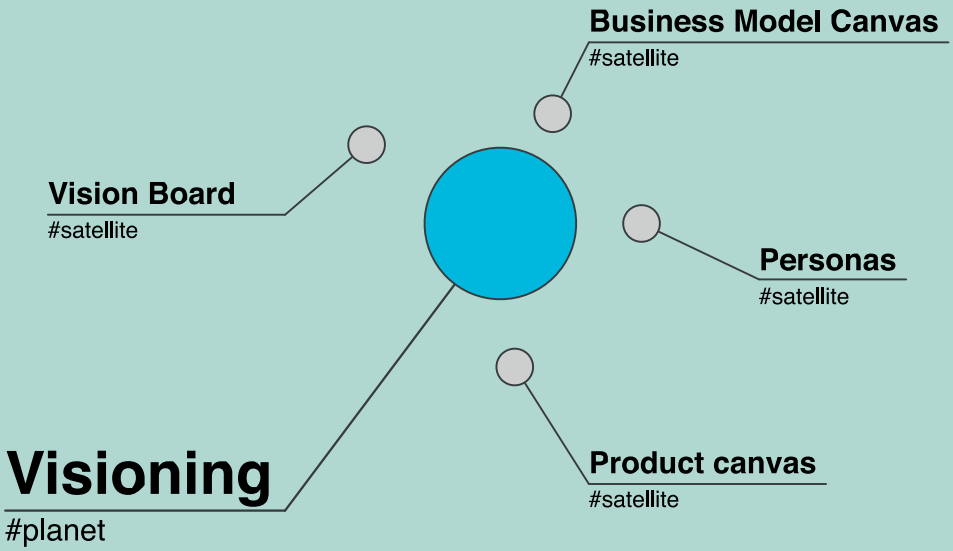

Ma sul sistema Inception vedremo come utilizzare alcuni strumenti quali lo story mapping...


...e le impact maps.
ACTORS
IMPACTS
WHY?
WHO?
HOW?
WHO?
HOW?
WHAT?
WHAT?
DELIVERABLES



Sono tutti strumenti utili per completare l'ultimo passaggio del processo, ossia quello che mette in fila le necessità di business dell'utente finale e le trasforma negli elementi grezzi di un backlog

Capitolo 1
Impostare la visione di prodotto



Introduzione

In questa parte del nostro libro sull'esplorazione della "galassia agile" parleremo di come **tradurre** un'**idea** in una serie di **funzionalità** da **implementare** per realizzare il prodotto. Parlando in termini comunemente utilizzati in ambito Agile, vedremo quali sono i passi per convertire una **vision** in un **product backlog** contenente elementi pronti per essere implementati dal team.

Questo argomento spesso viene trattato con titoli differenti, come **Lift Off**, **From Vision to Backlog** e altri ancora.

Questa fase è estremamente importante e delicata, dato che condiziona non solo **quello** che il team andrà a realizzare ma anche **come** lo realizzerà. Come responsabili del prodotto vogliamo evitare l'errore di costruire un prodotto che non soddisfi le necessità dei nostri utenti finali.

Prima di capire le dinamiche e le pratiche per concretizzare il **product backlog** è bene definire cosa si intende per **prodotto**, **servizio** e **progetto**.

Prodotto, servizio e progetto

Un **prodotto** è un "manufatto" che soddisfa uno o più bisogni del suo utilizzatore o risolve uno o più problemi.

Un **servizio** è l'equivalente di un prodotto, ma in senso immateriale. Anch'esso soddisfa uno o più bisogni o risolve uno o più problemi del suo utilizzatore.

Un **progetto** è un'attività limitata nel tempo, che ha un inizio e una fine, il cui scopo è creare prodotti e servizi di valore per gli utilizzatori.

In questo capitolo e nei successivi utilizzeremo i termini **prodotto** e **servizio** come sinonimi, visto che si tratta di due realtà ("materiale" e "immateriale") abbastanza sovrapponibili.

Invitiamo però il lettore a **non** confonderli con il **progetto**. La distinzione è molto importante perché permette di definire bene anche i ruoli all'interno del team. Il **progetto non ha valore** per l'**utilizzatore**: noi **non** guidiamo i piani di progetto di un'automobile ma, l'automobile vera e propria (il **prodotto**) che è il risultato di quel progetto.

Questioni agili

In un'ottica agile, questa distinzione è molto importante: l'approccio agile ha l'obiettivo di creare prodotti di valore, ossia soddisfare le necessità di business dell'utente finale, con il minor sforzo progettuale. Un prodotto non nasce dal nulla, ma si sviluppa grazie a un **processo graduale** che porta dall'identificazione di un bisogno o di un problema da risolvere, poi all'idea per la sua risoluzione, e infine alla realizzazione del prodotto o del servizio che soddisfa il bisogno o risolve il problema.

L'obiettivo di un qualsiasi sviluppatore di prodotto è riuscire a creare questi prodotti velocemente, a basso costo e generando il maggior valore possibile. È questa la sfida che ci poniamo quando affrontiamo un progetto. Vediamo di seguito i principali passi da svolgere per cercare di ottenere questo obiettivo.

Bisogni e problemi

Quando ideiamo, progettiamo e realizziamo un prodotto, dobbiamo tenere sempre presente che il risultato del nostro lavoro andrà a soddisfare i **bisogni** degli utenti o risolverà i loro **problemi**. Si parla spesso di attenzione al cliente e di prodotto "pensato attorno al cliente": purtroppo spesso questi buoni propositi si perdono per strada durante l'esecuzione del progetto, finendo per dare maggior rilievo ad aspetti ritenuti più importanti come le **scadenze** e il **budget**, **sottovalutando** invece di definire in modo approfondito cosa effettivamente crea **valore** per l'**utente** finale.

In Agile, volendo porre l'utente e i suoi bisogni al centro, si preferisce identificare fin da subito chi è l'**utente** a cui ci vogliamo rivolgere e quali sono i suoi **bisogni** che il prodotto andrà a soddisfare. Questi saranno i punti fermi durante tutto il progetto. Dovranno essere convalidati e verificati **costantemente** durante tutta la fase di implementazione del prodotto; qualora ci si accorgesse che il prodotto che si sta creando non risolve i bisogni identificati inizialmente, oppure che tali bisogni sono cambiati, è importante apportare le necessarie modifiche o addirittura interrompere il lavoro.

Condividere l'idea di prodotto

Oltre alla necessità di costruire il prodotto in risposta ai bisogni dell'utente, un altro importante aspetto da tenere in considerazione durante la fase di progettazione e realizzazione è creare una **visione condivisa** fra i vari *stakeholder* di progetto: il business, il team di sviluppo, il cliente o committente, gli esperti di dominio e, non ultimo, l'utente finale, quando è possibile coinvolgerlo direttamente. In tal senso è molto importante imparare a **condividere** l'idea di prodotto che abbiamo, renderla comprensibile a tutti i partecipanti al progetto in modo che lavorino tutti nella stessa direzione.

In questo capitolo parleremo di una tecnica che sfrutta alcuni **Canvas** e che è stata proposta la prima volta da Roman Pichler [1]. Si tratta di un'ottima fonte di ispirazione che poi abbiamo usato effettivamente nei progetti per descrivere la **vision** e il **backlog** dei diversi prodotti.

Sono passati ormai alcuni anni da quando abbiamo cominciato a usarla, sia nei corsi che durante le prime fasi di analisi con i clienti, e il riscontro sul campo ha dimostrato che si tratta di uno strumento sempre molto utile ed efficace.

Un grosso rischio: non condividere l'idea

Di solito, alla partenza del progetto, qualcuno — il **Project Manager**, il **Product Manager** o un'altra figura di rilievo — prepara una **presentazione** che descrive la **vision** del prodotto che si realizzerà e la presenta alle persone interessate per condividerne i punti e raccogliere i pareri.

Tipicamente, dopo questa presentazione, si procede a raccogliere i vari commenti e feedback per fare un **riassunto** delle **opinioni** e dei **consigli**. Tale report viene poi salvato da qualche parte e inviato ai partecipanti; di rado avviene un secondo meeting dove si prova a **rivedere** la **vision** sulla base dei commenti raccolti.

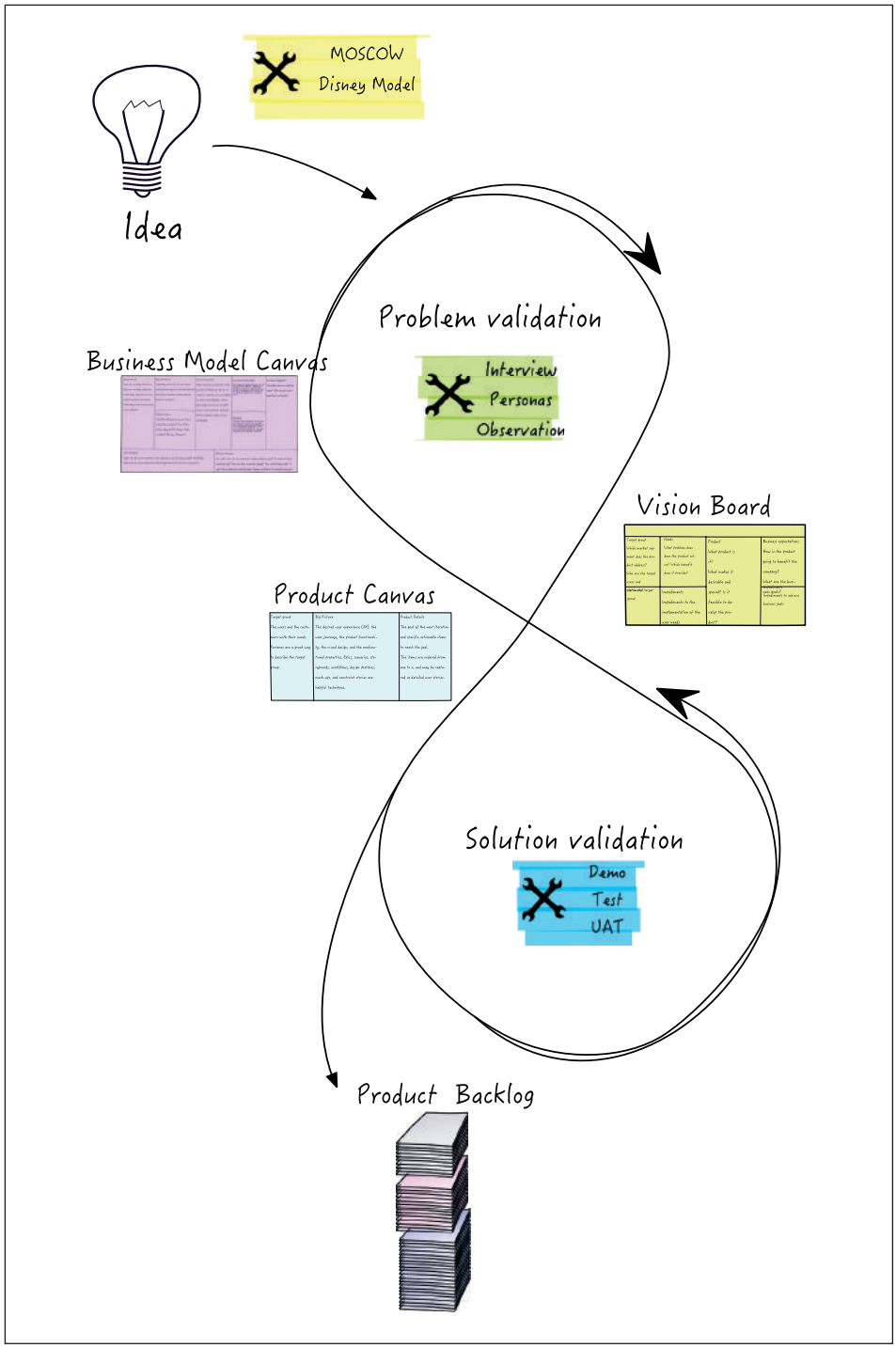


Figura II.1. Le fasi di envisioning di prodotto: dall'idea al product backlog.

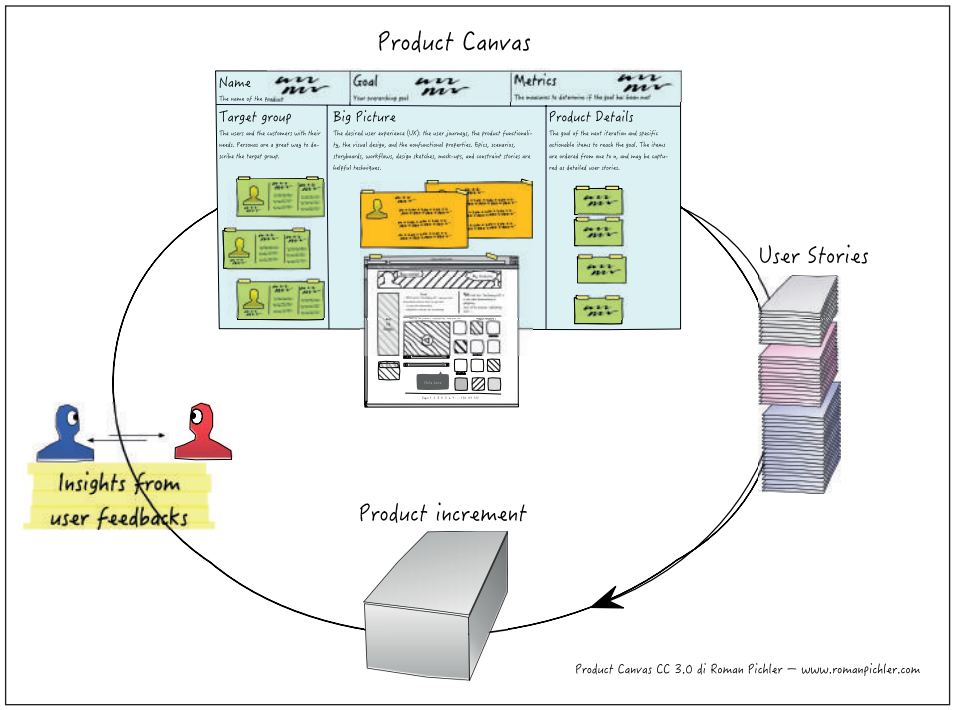


Figura II.2. Ciclo di miglioramento continuo del Product Backlog con il Product Canvas.

Questo è un metodo piuttosto classico di lavorare, diffuso in aziende e gruppi di vario tipo. Il limite principale di tale approccio risiede nello **scarso** livello di **allineamento** e condivisione. Effettuare un solo raffinamento spesso dà luogo a molti fraintendimenti o a passaggi non chiari: durante la fase di realizzazione del progetto questo è fonte di cambio dei requisiti, che risultano in conflitto con l'idea iniziale e non raramente anche fra di loro.

Arrivano i ritardi, il progetto che non risponde alle esigenze di business, il team non capisce la direzione impostata dal business e il business non capisce come mai il team ci metta così tanto tempo a fare le cose: in fondo si chiedono loro solo delle piccole modifiche.

Una delle cause principali risiede proprio nel non avere una **visione condivisa** e **compresa** veramente **da tutti**, perché non è stata fatta propria da coloro che lavoreranno al progetto. I **documenti non** sono stati **letti** con attenzione e le **riunioni** sono state seguite **distrattamente**.

Più coinvolgimento, più condivisione

Un buon modo per risolvere questi problemi, oltre a migliorare l'aspetto di comunicazione e condivisione, è quello di **coinvolgere** gli stakeholder non solo alla presentazione ma ben prima, durante la fase di definizione della **visione** di prodotto.

Utile in questo caso è quindi trovare un format di lavoro **collaborativo** semplice e rapido che mette tutti gli attori nella stessa stanza per un periodo breve ma molto efficace. Nelle prossime pagine vedremo il nostro approccio per risolvere questo problema tramite un workshop in cui i vari stakeholder lavorano fianco a fianco per un periodo breve ma molto intenso.

Costruire una vision condivisa: i vari passi del workshop

Nel workshop che abbiamo ideato, partendo da un'**idea**, una **frase**, un "**titolo**", si procede nel descrivere la vision **completa** e **coerente**, condividendola da subito con tutte le persone coinvolte nel progetto.

Per questo scopo si possono usare vari strumenti, come l'Elevator Pitch oppure cose più fantasiose come la **product box** o la classica ma sempre efficace **Vision Board**, strumento con il quale si sintetizzano gli **aspetti principali** del prodotto.

Il passo successivo prevede di elencare i **fattori portanti** del prodotto dal punto di vista del **business**. In questa fase si può utilizzare il **Lean Canvas** o in alternativa il **Business Model Canvas**.

Infine tramite l'utilizzo dello strumento delle **Personas**, si procede a individuare gli attori/utenti del sistema elencandone i relativi bisogni di business. Questo elenco porterà, per esempio tramite una sessione di **User Story Mapping**, alla creazione delle funzionalità da implementare o meglio dei bisogni da risolvere: si creerà quindi il **Product Backlog**.

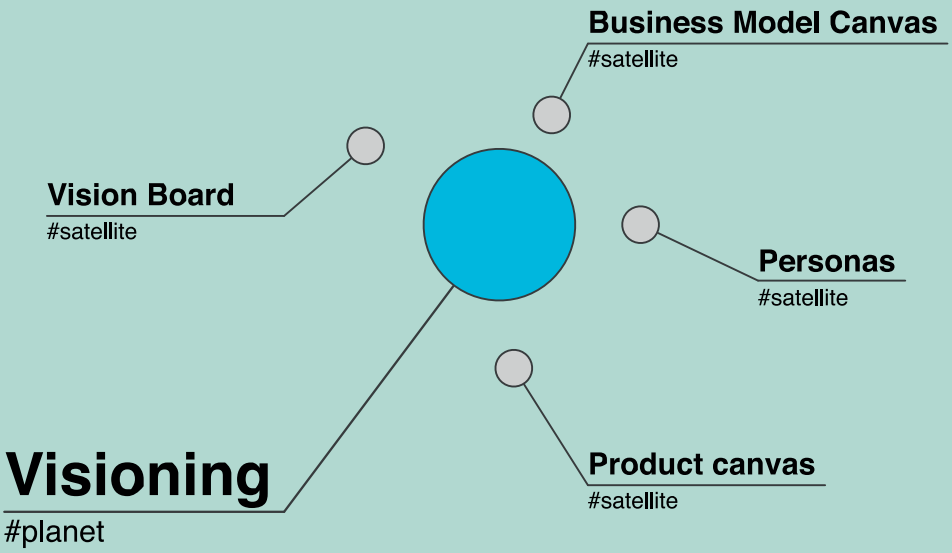
Le storie del **Product Backlog** verranno poi **raffinate** grazie a iterazioni successive e alla verifica del prodotto incrementale confrontando i "viaggi" degli **utenti reali** con quelli delle **personas** pensate durante il workshop.

La cosa importante della **vision** basata su questi tre canvas è che sia sempre coerente e allineata con il mondo reale che sta usando il nostro prodotto.

Riferimenti

Roman Pichler, *The Product Vision Board*, maggio 2011
http://www.romanpichler.com/blog/agile-product-innovation/the-product-vision-board

Capitolo 2
La Vision Board



Introduzione

Cominciamo quindi a fare i primi passi utilizzando uno degli strumenti a nostra disposizione: vedremo in questo capitolo in cosa consiste una **Vision Board** e quanto il suo corretto utilizzo possa aiutare a mettere a fuoco la visione di prodotto. Si tratta di uno strumento semplice ma decisamente efficace.

Una lavagna per favorire la visione

L'obiettivo della **Vision Board** è di catturare i **quattro aspetti** fondamentali di un prodotto: gruppo target di **utenti**, **bisogni** di business, **proprietà** del prodotto e proposta del prodotto in termini di **valore** per l'azienda.

Target Group

Indica a chi è rivolto il prodotto tramite l'elencazione delle categorie di utilizzatori finali del prodotto che state pensando. È utile non essere troppo dettagliati ma identificare le macro-categorie. Man mano che la visione sarà completata, si potranno far emergere nuove categorie oppure aggregarne di esistenti.

User needs

È l'elenco dei bisogni di business, ad alto livello, dei macro utenti o quali problemi va a risolvere il prodotto. Durante la compilazione si deve cercare di identificare almeno

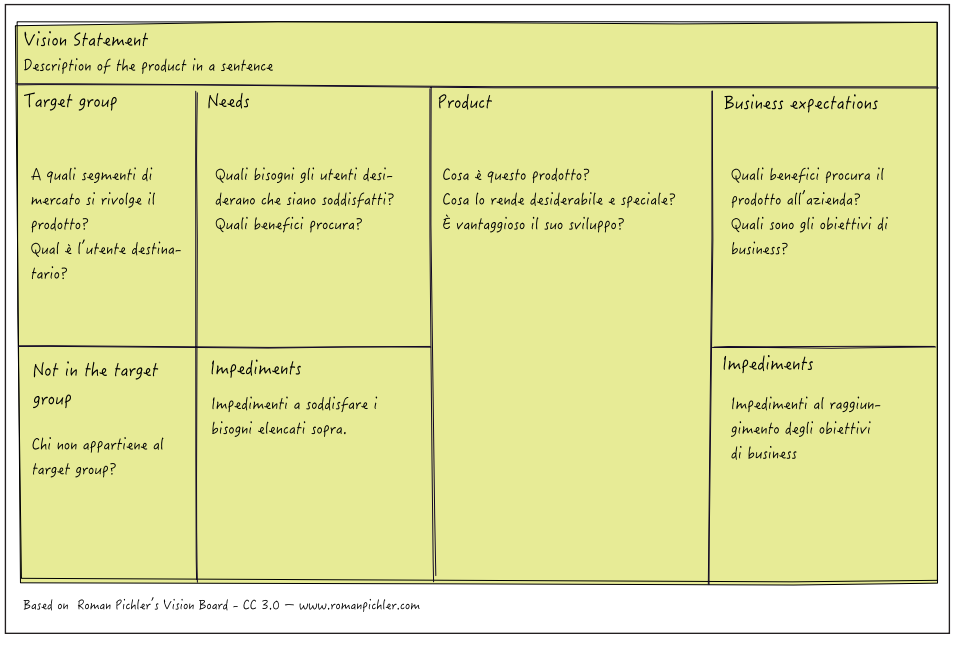


Figura II.3. Vision Board, con le colonne per "gruppi bersaglio", "bisogni", "caratteristiche di prodotto" e "aspettative di business".

un bisogno di business per ogni gruppo bersaglio. Nel caso il bisogno sia sentito da più target group e un gruppo bersaglio non abbia un bisogno unico, si può prendere in considerazione di aggregare i target group. È importante essere sintetici nella Vision Board.

Product properties

Questa sezione indica quali sono le caratteristiche principali e di maggior rilievo del prodotto; è una applicazione web? Mobile? Prevede la sincronizzazione in tempo reale con il cloud?

In questa parte è bene limitarsi a descriverne pochi, 3, 4, massimo 5 è una buona metrica. In questa lista ristretta, non si devono elencare quelle caratteristiche scontate ma si deve tentare di descrivere quelle di maggior rilievo.

Value Proposition

In questa parte sono riportate le aspettative di business che l'azienda pone nel prodotto che sta realizzando. Le aspettative possono essere sia di incremento di ricavi sia di altra natura come risparmio di costi di esercizio, mantenimento della quota di mercato, miglioramento della soddisfazione dei propri clienti.

Vision Statement

Le informazioni contenute in queste quattro colonne sono poi riassunte nel **Vision Statement**, una **frase** che tutte le persone coinvolte nel progetto dovrebbero conoscere e fare propria. In figura 3, abbiamo riportato un esempio di Vision Board.

Uso della Vision Board

Metaforicamente, la vision è il **faro guida** del progetto: se questo faro si spegnesse o venisse perso di vista, il progetto stesso potrebbe subire dei danni o dei cambi di rotta che potrebbero portare al suo fallimento.

La **sintesi** della vision aiuta a mantenere i concetti semplici da condividere, da memorizzare e da verificare costantemente. L'ideale sarebbe avere la **Vision Board** sempre a portata di sguardo, sia per il team che per il business, insieme agli altri **information radiators** ("diffusori di informazione").

Un esempio di Vision Board

A scopo di esempio, immaginiamo come si costruirebbe la Vision Board di un ipotetico servizio di home banking.

Per la compilazione della board, in genere si procede inserendo le informazione relative nelle quattro colonne della **Vision Board** da sinistra verso destra; al termine si prova a sintetizzare il tutto in una frase rappresentativa dei quattro fattori che compongono la vision.

Nel caso dell'esempio, il **Vision Statement** potrebbe essere: "Per chi vuole gestire il proprio conto corrente velocemente, pagando solo per i servizi attivati, **MyHomeBank**

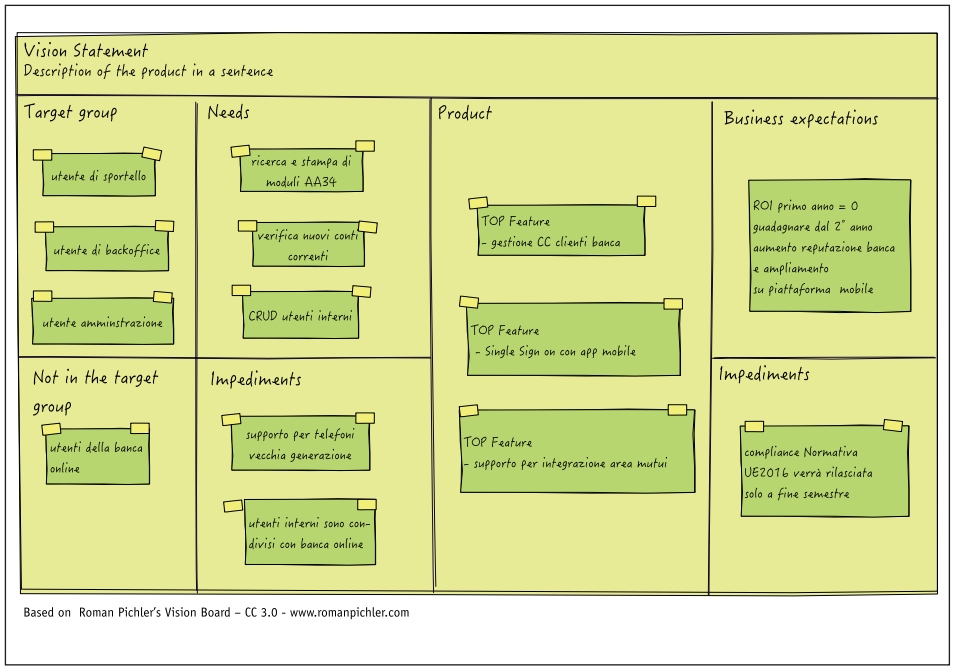


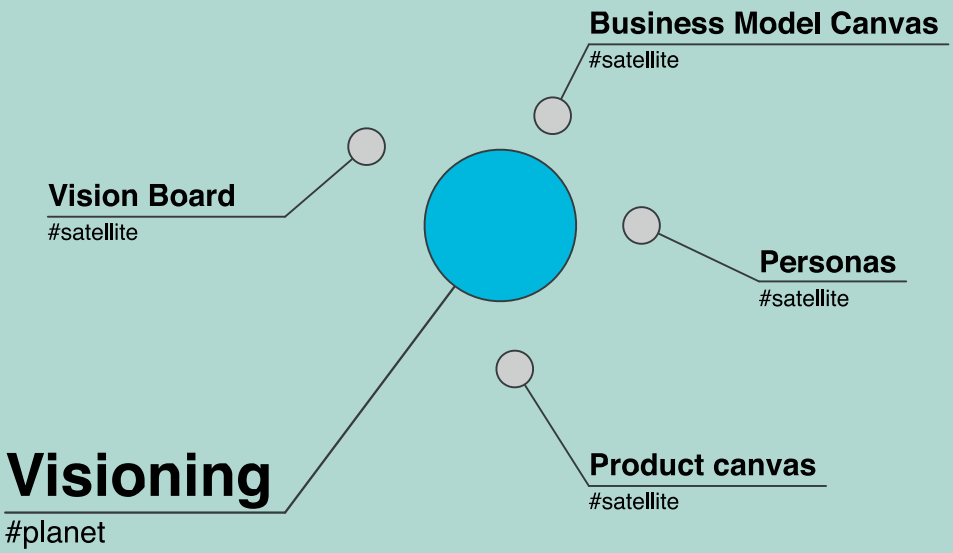
Figura II.4. Esempio di Vision Board per un ipotetico servizio di home banking.

è l'app mobile che ci permette di aumentare il numero di utenti". Si noti la parte "... che **ci** permette di aumentare...": in questo caso il plurale (il "**ci**") sta a significare "**noi**, **azienda** che commercializza questo servizio".

È questo un aspetto molto importante da sottolineare: troppo spesso, infatti, viene trascurato il **motivo aziendale** per il quale il prodotto è realizzato. È bene invece esplicitarlo ed evidenziarlo bene. Il motivo aziendale sarà infatti molto importante quando si dovrà procedere all'ordinamento degli elementi del **Product Backlog**.

Una volta che la Vision Board sia stata completata e sintetizzata in una frase esplicativa, è utile lasciarla "sedimentare" un po' di tempo, per poi rivederla tutti insieme in modo da **verificarla** e consolidarla. Una volta rivista e migliorata, la **Vision Board** sarà pronta e a disposizione di tutti, e si potrà passare alle fasi successive quando si useranno altri due canvas molto importanti: il **Business Model Canvas** e poi il **Product Canvas**.

Capitolo 3
Validare il modello di business della vision



Lean Canvas

Per quanto buona e ben descritta, una vision da sola non basta per creare un prodotto: affinché sia realizzabile, occorre legarla a un **modello** di **business**.

Una possibile risposta è lo strumento **Lean Canvas** [1] ideato da Ash Maurya in *Running lean* [2] e sviluppato a partire dal **Business Model Canvas** [3] di Alexander Osterwalder.

In breve, il **Lean Canvas** è uno strumento utilizzato principalmente per creare velocemente un modello di business. A differenza di un business plan, tipicamente composto da oltre 30 pagine e che necessita di settimane o mesi per essere redatto, il **Lean Canvas** è composto da un'unica pagina realizzabile anche solo in 20 minuti.

È quindi uno strumento **veloce** e conciso che permette, da un lato, di produrre e validare in poco tempo molteplici modelli di business e, dall'altro, **obbliga a focalizzarsi** esclusivamente sui punti salienti e più importanti del prodotto.

Come è fatto un Lean Canvas

Il **Lean Canvas** è composto da **9 sezioni** che possono essere compilate in ordine sparso, man mano che si hanno le informazioni, anche se tipicamente si segue l'ordine seguente:

- customer segmentation
- problem

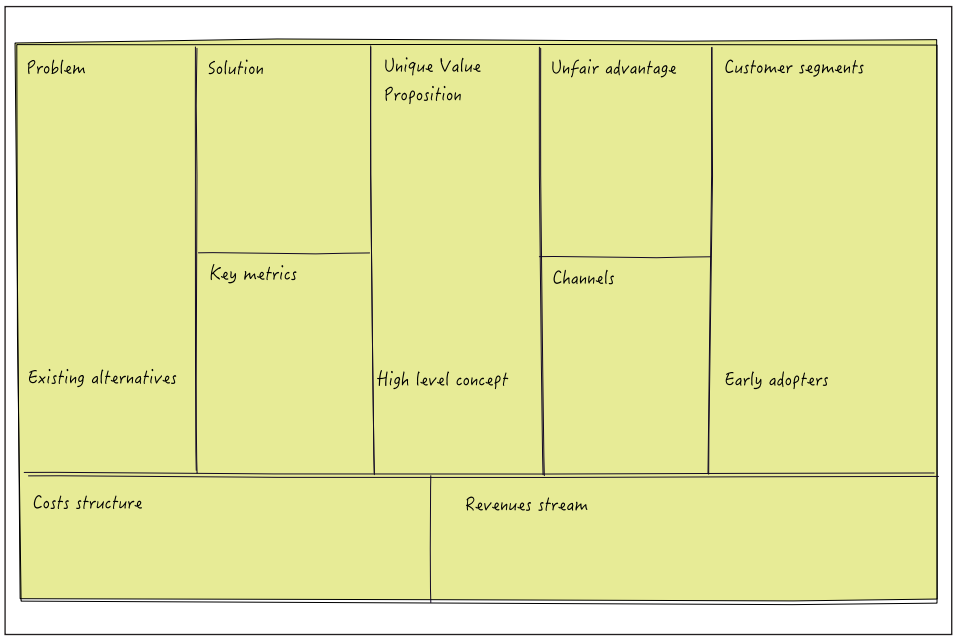


Figura II.5. Lean canvas, con le diverse sezioni che lo compongono.

- unique value proposition
- solution
- channel
- cost structure
- revenue stream
- key metrics
- unfair advantage

Customer segmentation

È difficile costruire un prodotto/servizio se non conosciamo **chi** sono i nostri clienti e i nostri utenti e come questi sono "segmentati": stile di vita, età, amatori o professionisti del settore, livello di coinvolgimento tecnologico e così via. In questa sezione andiamo principalmente a sciogliere questi nodi.

È bene porre attenzione alla differenza tra **utenti** e **clienti**: mentre i primi sono quelli che **utilizzano** il nostro prodotto/servizio, i secondi sono quelli che lo **pagano**. In alcuni casi utenti e clienti coincidono, in altri no.

Si possono eventualmente valutare anche le caratteristiche dell'utente "ideale" che userà il prodotto o il servizio fin da subito, nello spazio **early adopters**.

Problem

Citando un detto di Charles Kettering, "un problema ben identificato è un problema mezzo risolto". In questa sezione si vanno a individuare i principali **3** problemi che deve risolvere chiunque dovrà realizzare il prodotto.

Se esistono, è consigliato anche elencare le possibili soluzioni del problema che ad oggi già esistono, anche se si tratta di soluzioni parziali. A tale scopo è presente lo spazio per le **existing alternatives**.

Unique value proposition

La UVP è una **frase** che sostanzialmente spiega il motivo di esistere del prodotto/servizio che si sta realizzando. In che cosa si differenzia dagli altri? Perché i clienti dovrebbero comprarlo?

Questa forse è la sezione più importante del modello in quanto catalizza l'attenzione del lettore e illustra le **motivazioni** alla **base** della **vision**.

È possibile inoltre specificare un concetto di astrazione ad alto livello (spazio **high level concept**) che serva a spiegare ulteriormente la **unique value proposition**: l'esempio tipico è quello che ci spiega come "Netflix = Spotify dei film".

Solution

Specularmente alla sezione **problem**, qui si elencano le principali **3 caratteristiche** del prodotto che risolvono i problemi sopra esposti e che dimostrano la **unique value proposition**.

Channel

In questa sezione si vanno a elencare i **canali** per mezzo dei quali si può "consegnare il valore del prodotto" ai clienti. È molto importante capire **quanti** e **quali** sono questi canali, in quanto la loro mancata identificazione è una delle principali cause di fallimento.

Costs structure

La sezione relativa alla **costs structure** serve a identificare quali saranno i **costi** da sostenere per realizzare il modello di business descritto. La **struttura dei costi** deve tenere in considerazione svariati aspetti:

- identificare la **frequenza** dei costi;
- stimare l'**ammontare** dei costi;
- individuare i costi **fissi**, ossia quelli che rimangono invariabili indipendentemente dalla quantità prodotta;
- individuare i costi **variabili**, ossia quelli che sono funzione della quantità prodotta e che variano al crescere di quest'ultima;
- individuare possibili **economie**: di **scala** in cui il costo si riduce all'aumento della quantità prodotta, di **esperienza**, in cui il costo diminuisce al crescere dell'esperienza accumulata nel realizzare il prodotto, di **scopo**, in cui il costo diminuisce grazie a produzioni congiunte e al riutilizzo di conoscenze, e così via.

Revenues stream

In contrapposizione alla struttura dei costi, qui si va invece a identificare il **flusso dei ricavi** generato dalla realizzazione del modello di business. Più in particolare in questa sezione si andrà a descrivere la struttura dei prezzi.

Punto focale resta la definizione del **prezzo** e, per farlo, potrebbe essere utile considerare i seguenti elementi:

- la **strategia** di prezzi da utilizzare, ad esempio **freemium** (una parte del prodotto/servizio gratuita e un'altra a pagamento), **ads** (ricavi generati da pubblicità), **subscription** (ricavi generati da quote di iscrizioni/abbonamenti), **licensing** (ricavi derivanti da vendita di licenze), e così via;
- le **modalità** di pagamento (a **rate**, **una tantum**) e così via;
- il **posizionamento** del prodotto/servizio nel mercato.

Key metrics

In questa sezione vanno elencate tutte le attività che si andranno a **misurare per capire** l'andamento del business. Esempi di queste **metriche** sono: **acquisizioni**, **attivazioni**, **retention**, **referenze**, **revenue**, e così via.

Unfair advantage

Jason Cohen descrive un **unfair advantage** come "qualcosa che non può essere facilmente copiato o comprato". Molte volte, individuare gli **unfair advantage** può essere

difficoltoso e, almeno nella fase iniziale non è sbagliato lasciare questa sezione del canvas in bianco, come suggerisce Maurya.

Per ora i lettori non si preoccupino eccessivamente di questo aspetto, tanto ci torneremo adeguatamente sopra nei prossimi capitoli, in cui vedremo l'uso del Lean Canvas in maniera più dettagliata.

Conclusioni

Il Lean Canvas è certamente uno strumento che aiuta nell'affrontare un argomento complesso e importante, come quello dell'individuazione degli aspetti di business del prodotto che si deve realizzare; questo canvas si rivela molto utile per la sua capacità di focalizzare l'attenzione sulla definizione di alcune caratteristiche del prodotto nelle fasi iniziali.

Su di esso, e sul suo uso pratico, torneremo nei prossimi capitoli. La cosa fondamentale da capire è che un prodotto o un servizio devono nascere come risposta a una serie di bisogni o di problemi — siano essi preesistenti o indotti — che un utente può avere.

Una volta identificate le possibili soluzioni e il modo in cui sostenerle, arriva il momento in cui dobbiamo condividere e comunicare la vision del prodotto. Nel prossimo capitolo vedremo quali strumenti abbiamo a disposizione per farlo.

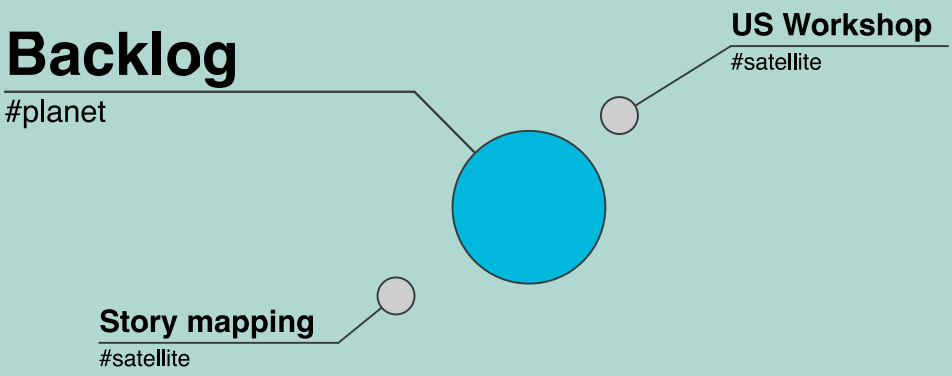
Riferimenti

Ash Maurya, *Why Lean Canvas vs Business Model Canvas?*
https://goo.gl/iHqTLV

Ash Maurya, *Running Lean: Iterate from Plan A to a Plan That Works*. O'Reilly, 2nd edition, 2012

Business Model Canvas
http://www.businessmodelcanvas.it

Capitolo 4
Raccogliere le storie con la User Story Map



Introduzione

Una celebre frase di Kent Beck recita: "Le roadmap dei prodotti dovrebbero essere liste di domande e non elenchi di funzionalità". In quest'ottica, nel presente capitolo intendiamo parlare delle tecniche per arrivare a definire l'**elenco delle cose da fare**.

In **Scrum**, questo elenco di cose da fare è detto **Product Backlog Items**. Per arrivare a una prima versione del **PBI**, è necessario individuare la lista di bisogni e di domande a cui deve rispondere il sistema e trovare anche un **ordine** con cui metterle in fila. Fra le varie tecniche che si possono utilizzare, la **User Story Mapping** è una delle più efficaci e, proprio per questo motivo, la sua diffusione è sempre più ampia.

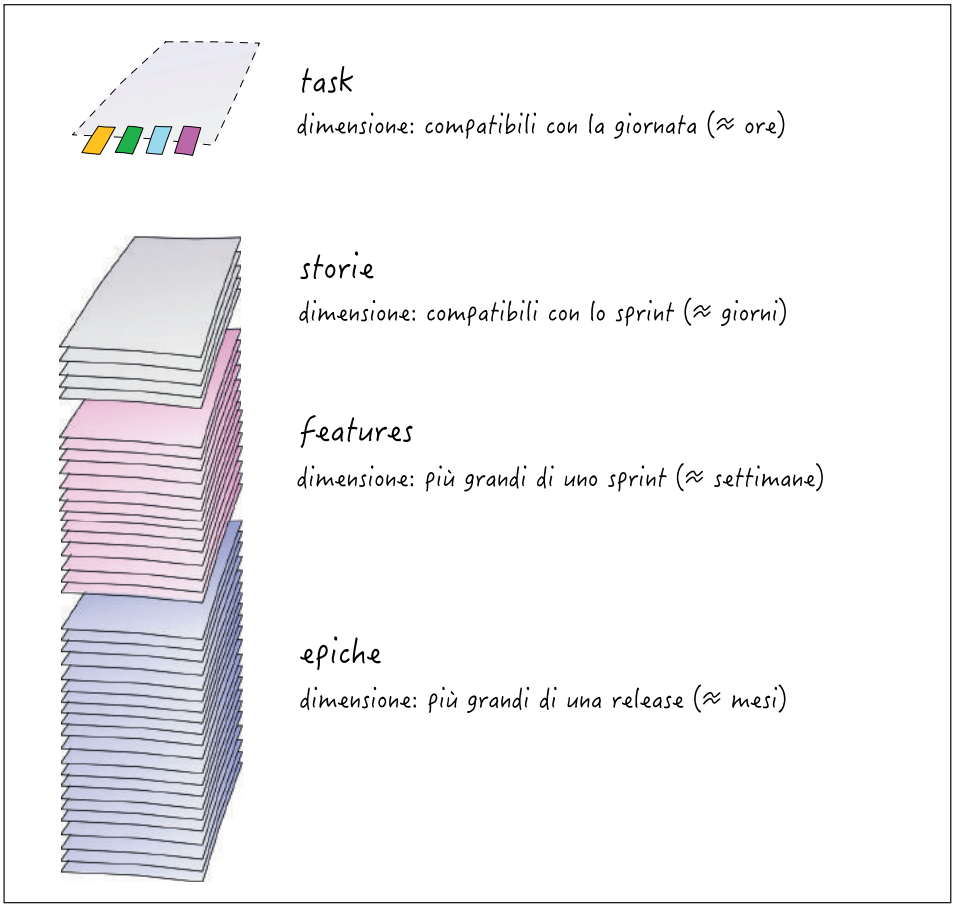


Figura II.6. Il Product Backlog di un progetto Scrum; contiene gli elementi che devono essere implementati per la realizzazione del prodotto. Sono ordinati per priorità (o valore) e sono disposti in modo che gli elementi in alto siano piccoli a sufficienza per poter essere messi in lavorazione, mentre in basso si trovano gli elementi più grandi. Normalmente il formato utilizzato è quello delle storie utente che diventano epic, feature, user story.

Come è fatto un Product Backlog

Il **Product Backlog** è una **pila** che contiene l'**elenco** dei **bisogni** dell'utente che utilizzerà il prodotto ideato e da realizzare. Nella maggior parte dei casi, gli elementi sono descritti con il **formalismo** delle **storie utente**, ma si possono usare anche altri formati.

Nella parte alta di questa pila ci sono gli elementi ad **alta priorità**, quelli che si ritiene sia più utile realizzare per primi perché la loro implementazione fornisce all'utente un valore maggiore; nella parte bassa, invece, ci sono le funzionalità che sono spesso considerate "nice to have" ossia aggiunte utili al prodotto, ma **non indispensabili** per l'utilizzo.

Contenuto del backlog e ordine delle storie

Arrivare a definire il **contenuto** e l'**ordine** del backlog è probabilmente una delle attività più importanti di un progetto agile basato su Scrum o su altra metodologia. Lo Story Mapping aiuta esattamente in questo compito: tramite un lavoro di gruppo, una sorta di discussione collaborativa, si arriva in genere non solo alla definizione del contenuto del **Product Backlog**, ma anche alla realizzazione di una **mappa** delle **cose da fare.**

Tale mappa, di fatto una sorta di **todo list** bidimensionale, rappresenta spesso un valido strumento **complementare** o **in sostituzione** del backlog, proprio grazie all'aggiunta di una seconda dimensione, garantendo una visione d'insieme spesso migliore di quella offerta dalla lista **monodimensionale del backlog**.

Avere a disposizione una visione bidimensionale del lavoro da svolgere agevola inoltre le attività di valutazione, prioritizzazione e raffinamento delle storie. Grazie alla sua immediata capacità di fornire una rappresentazione **visuale** delle cose da fare, la **Story Map** definisce quindi una sorta di linguaggio universale facilmente comprensibile anche da persone non facenti parte del team di sviluppo.

Il processo stesso di creazione e di gestione della mappa agevola la **discussione collaborativa** del gruppo; la tecnica di lavoro, iterativa e incrementale, si basa su un procedimento semplice e comprensibile anche da personale non tecnico o che non sia stato introdotto alle pratiche agili: per questo spesso viene utilizzata come strumento alternativo nelle sessioni di **project planning tradizionale**, dove sia necessario chiarire gli aspetti legati alle scadenze, alle tempistiche e allo scope delle varie release di progetto/prodotto.

Come nascono le Story Maps: le considerazioni di Jeff Patton

Anche se le **story maps** sono utilizzate da molto tempo in varie forme, il primo che probabilmente le ha formalizzate e rese famose è stato Jeff Patton, che l'ha descritta nel suo libro *User Story Mapping* [1]. Fra le molte caratteristiche che secondo Patton rendono importanti le mappe, è la loro capacità di offrire una **rappresentazione persistente** del lavoro di **visioning** nonchè del contesto di lavoro.

Nel suo libro infatti Patton racconta:

«Trascorriamo molto tempo a lavorare con i nostri clienti e ci sforziamo di comprendere i loro obiettivi, i loro utenti e le parti principali del sistema che potremmo costruire. Infine arriviamo ai dettagli: le diverse parti delle funzionalità che ci piacerebbe costruire. Io immagino questo processo come se fosse un albero: il tronco è costituito dagli obiettivi o dai vantaggi desiderati per cui si decide di costruire il sistema; i rami grandi principali rappresentano gli utenti; i rami secondari più piccoli sono le capacità di cui essi hanno bisogno; infine, le foglie sono le storie utente, sufficientemente piccole da poter essere impiegate nelle iterazioni di sviluppo.
Dopo tutto questo lavoro, in cui si è creata tutta questa conoscenza condivisa, a me sembra che spesso si comincino a staccare le foglie dell'albero e a buttarle nei sacchi per lo smaltimento, e poi si proceda ad abbattere l'albero. Ecco come mi appare un backlog piatto: un sacco di pacciame senza un contesto [...]
Le storie davvero grosse possono essere considerate come delle 'epiche': Mike Cohn le descrive per l'appunto in questo modo. Sono storie, solo che si tratta di storie veramente grandi: troppo grandi per essere valutate e sviluppate. Quando un'epica entra nel nostro backlog e arriva il momento di discuterla nel dettaglio, vedo spesso le persone che la rimuovono dal backlog stesso, la scompongono e sostituiscono i pezzi che riescono a identificare. Questa è la classica situazione in cui non posso trattenere una smorfia quasi di dolore: è esattamente il caso dell'abbattimento dell'albero del quale si mantengono le foglie dentro un sacco per il pacciame. Infatti, quella storia grande costituiva comunque il contesto. Era il mio modo semplice di pensare a riguardo dell'intera attività che le persone stavano svolgendo; era il mio modo veloce di spiegare agli altri come vedo io il sistema».

A riprova che la **User Story Mapping** non è una tecnica inventata da Jeff Patton, c'è la stessa testimonianza dell'autore che racconta come spesso trovasse presso i clienti tecniche analoghe o del tutto simili. In questo senso Patton ci dice quindi che la **Story Mapping** non è una tecnica, ma piuttosto un **pattern** di lavoro:

"Se si spiega un concetto e si riceve come risposta 'Che bella idea!', non siamo di fronte a un pattern. Ma se le persone ci dicono 'anche noi usiamo qualcosa di simile a questo strumento', allora siamo in presenza di un pattern".

Story Map Workshop, il processo per arrivare alla mappa delle storie

In questo capitolo vediamo come si può arrivare alla definizione della mappa delle storie tramite un processo detto di **Story Map Workshop**: partendo dal lavoro descritto dallo stesso Jeff Patton nel suo libro, aggiungeremo alcune tecniche o variazioni che abbiamo sperimentato e trovato utili nel corso delle attività di coaching in progetti agili.

In particolare l'applicazione della tecnica del **buy an issue** per la valutazione del valore delle cose da fare è stata da noi introdotta — e siete quindi liberi di considerarla una stupidaggine — prendendo spunto da tecniche di **Value Stream Mapping** e **serious gaming**; si veda a tal proposito [2], al paragrafo intitolato "A tasty cupcase".

In linea con la filosofia agile, la realizzazione della mappa viene fatta dalle stesse persone che poi effettueranno la lavorazione del progetto: essenziale la presenza del **Product Owner** — o figura analoga se il progetto non sarà organizzato secondo Scrum — e di qualche membro del **team** di **sviluppo** oltre agli altri **stakeholders** interessati allo sviluppo del prodotto. In questa prima fase non è essenziale che partecipi il team di sviluppo al gran completo: essere in pochi può risultare un buon modo per mantenere un clima intimo e riflessivo. Molto utile, in alcuni casi necessaria, la presenza di un **facilitatore**, un coach che guidi la riunione, scandisca i tempi e definisca le attività.

Elenco degli utenti del sistema

Per identificare un primo **elenco** con le **macrofunzionalità**, si cambia la prospettiva, guardando il sistema dal **punto** di **vista** degli **utenti** e immaginando quindi quali saranno le **attività** che essi possono o desiderano svolgere all'interno del prodotto.

Per prima cosa quindi il gruppo di lavoro procede nell'individuare l'elenco degli **utenti** del sistema: questo lavoro può essere fatto in modo collaborativo (tutti di fronte alla lavagna) oppure isolatamente, tramite la tecnica del **silent brainstorming**, tecnica di collaborazione basata sull'uso di cartoncini attaccati alla parete. La tecnica del silent brainstorming è descritta in modo piuttosto chiaro da Steve Rogalsky [8]: in sintesi, potremmo dire che si tratta di una modalità di lavoro in cui ogni membro del team, lavorando **individualmente** e **in silenzio**, per prima cosa stila un proprio elenco

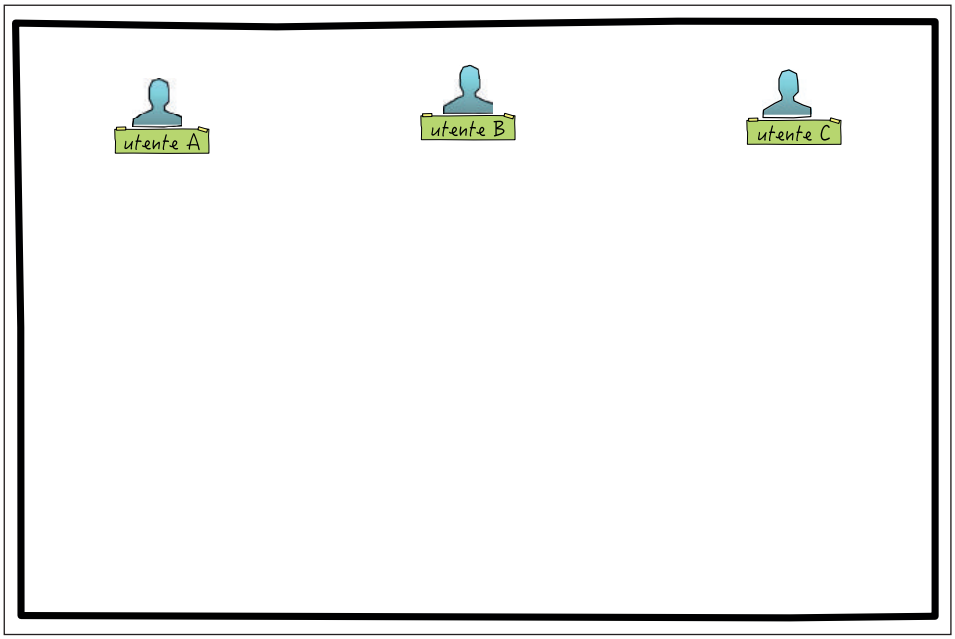


Figura II.7. Prima fase del lavoro: si stila l'elenco delle personas.

di risposte o proposte. Successivamente **ogni elenco** personale verrà **presentato** agli altri, **discusso** e **confrontato**.

Non sempre è possibile nè semplice procedere in questo modo. A volte è preferibile partire definendo l'elenco delle classi di utenti dell'applicazione: una tecnica molto utile in questo caso è certamente quella della definizione delle **Personas**, di cui si è avuto modo di parlare sia in alcuni articoli pubblicati su MokaByte [3] dedicati allo sviluppo dell'esperienza utente, che in un articolo [4] che è stato opportunamente rielaborato per essere incluso in questa parte del libro.

L'obiettivo di questa fase è quello di arrivare con un elenco di nomi di ruoli o personas scritti ognuno su un cartellino e attaccati nella parte alta della parete o lavagna, come riportato nella figura 7.

Le attività svolte dagli attori

Dopo aver individuato gli **utenti** del sistema, si procede a individuare i bisogni di business di ogni utente, in modo da individuare l'elenco delle operazioni che gli utenti svolgeranno utilizzando il prodotto, ovvero delle funzionalità che il prodotto dovrà implementare.

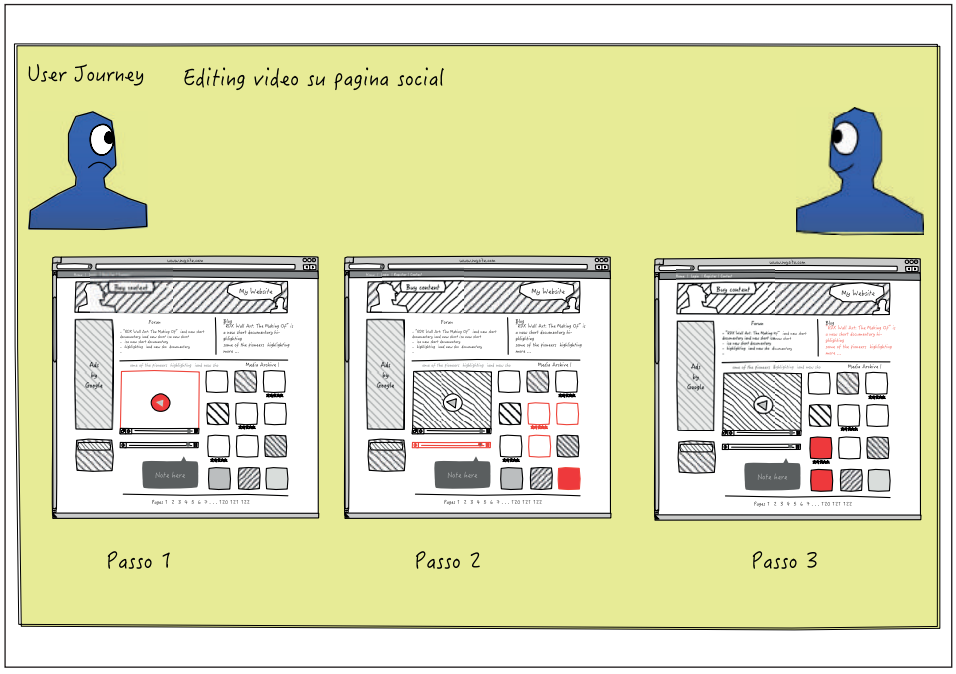


Figura II.8. Uno user journey: il viaggio inizia fuori dal prodotto con l'utente che ha un bisogno irrisolto. L'utente "entra" nel prodotto effettuando alcune operazioni. L'utente esce dal prodotto con il suo bisogno soddisfatto.

Anche in questo caso si può lavorare in modalità **silent brainstorming**, in cui ognuno scriva su dei biglietti un elenco di funzionalità: ogni bigliettino dovrebbe esprimere una bisogno di business preciso, quindi è utile identificare il **verbo correlato** e concludersi eventualmente con un **complemento oggetto**; per esempio: "**creare nuovo utente**", "**inviare mail**", "**stampare report**" e così via.

Dopo che ogni partecipante al workshop ha stilato la propria lista di bisogni, ogni persona presenta i propri cartellini uno per uno attaccandoli alla lavagna sotto l'utente corrispondente.

In alternativa, se il gruppo di lavoro è numeroso, si possono creare tanti piccoli gruppi, ognuno dei quali si concentra su un utente specifico, identificandone i bisogni di business.

Un formalismo molto utile per descrivere meglio il bisogno utente e come questo possa essere soddisfatto dal prodotto, è quello di rappresentare l'**esperienza utente** nel sistema durante l'interazione dell'utente con il sistema. Si prova a disegnare un miniflusso basato su questo schema:

- utente è triste perché ha un bisogno irrisolto
- utente inizia ad utilizzare l'applicazione seguendo una serie di passi
- utente ha soddisfatto il suo bisogno ed è contento

Facciamo un esempio: Marco è un consulente che svolge attività presso aziende del settore telecomunicazioni. Marco per promuovere la sua attività spesso pubblica sui social le sue attività lavorative, le conferenze a cui partecipa, i corsi che eroga.

Ogni volta che parla a una conferenza, Marco alimenta la sua rete di contatti, pubblicando su Linkedin, Twitter e Facebook un post con il titolo del suo talk, magari una foto e il link al sito della conferenza.

Il profilo di Marco e il suo bisogno quindi è piuttosto chiaro. Supponiamo che il progetto che si sta sviluppando, si prefigga di risolvere questa necessità, per esempio implementando un sistema multisocial che dallo smartphone permetta di pubblicare con un solo flusso di operazioni, sui vari social; oggi esistono diverse app che svolgono questo compito, ma non è importante, si ipotizzi che non ne esistano.

Marco è in procinto di partecipare a una importante conferenza del settore e quindi ha il bisogno di comunicarlo. Il flusso di operazioni di Marco potrebbero essere le seguenti:

- Marco ha un bisogno insoddisfatto e per questo è triste (la metafora è per rafforzare il bisogno) perché nessuno sa del suo talk;
- Marco prende l'app social che si sta progettando (ipotizzando che sia già realizzata);
- seleziona i social network su cui pubblicare il post;
- inserisce del testo;
- mette il link al sito della conferenza;
- allega una foto;
- associa degli hashtag;
- pubblica;
- Marco adesso è contento perché ha risolto il suo bisogno.

Questo flusso di operazioni spesso viene codificato tramite il formato degli **User Journeys**, un altro strumento preso in prestito dalla disciplina del Service Design.

Un accorgimento in più: la linea temporale

A mano a mano che tutti attaccano i propri cartellini, può essere utile che ogni persona disponga i propri in una sorta di **corsia orizzontale** cercando per quanto possibile di considerare la dimensione orizzontale come una sorta di **linea temporale** relativa alle operazioni che l'utente svolge all'interno del sistema (figura 9). Questa è una opzione che non si trova nel libro di Patton, ma che abbiamo sperimentato più volte e che può essere un valido ausilio per chiarire ulteriormente il flusso di interazione fra gli utenti e il sistema.

In questa fase i cartellini che sono organizzati in colonne rappresentano le funzionalità a supporto delle operazioni dei vari utenti. Sono possibili quindi dei duplicati, cosa che non costituisce un problema, potendo comunque mantenere un flusso *end to end*.

Attività utenti e colonna portante

Al termine di questo lavoro, ogni **colonna** formata dovrebbe rappresentare una macroarea funzionale del flusso *end to end*.

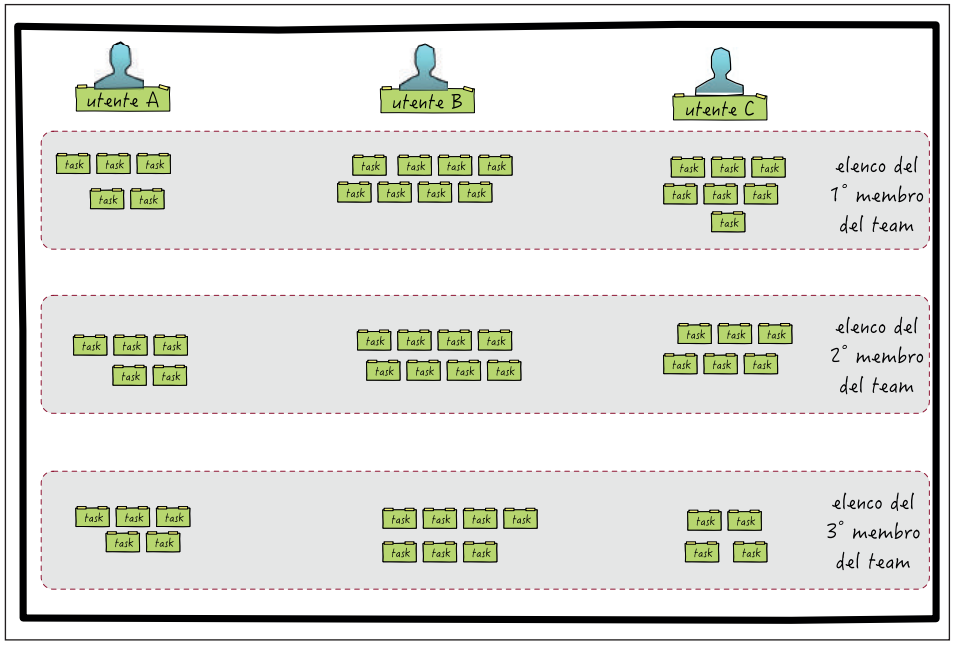


Figura II.9. Dopo aver individuato le personas del sistema, si elencano le azioni che ogni utente svolge all'interno dell'applicazione. Ogni componente del team attacca alla parete i propri biglietti in modo da formare una sorta di corsia orizzontale.

Figura II.10. La backbone della mappa.

Queste macro aree funzionali sono quelle che Jeff Patton chiama **User Activities**: l'insieme delle attività da luogo alla **Backbone** dell'applicazione.

Durante l'attività di raggruppamento per identificare le **User Activities**, si consiglia di disporre i vari cartellini nella parte alta della board a formare, se possibile, un'unica riga che rappresenta la raccolta di **funzionalità ad alto livello**, che verranno chiamate **User Tasks** (un'altra parola presa in prestito dalla teoria della UX).

Una visione comune sul valore delle attività

A questo punto, potrebbe essere utile analizzare se le varie persone (in particolare il **Product Owner** vs. il resto del team) hanno una **visione comune** circa l'importanza o il **valore** delle varie attività. Per fare questa valutazione, oltre a stimolare una discussione libera, si può utilizzare una tecnica presa in prestito dal **Value Stream Mapping**, che a volte viene presentata in forma di gioco dal nome **Buy An Issue**.

Supponendo di dare dei soldi **virtuali** ai partecipanti (potrebbero essere 100 € o 100 $ o semplicemente **100 punti**) si chiede a ciascuna persona di disporre i questi soldi sulle varie attività in funzione del **valore**, dell'**utilità** o genericamente del **ROI** (indice di redditività) che ci si attende dall'implementazione di tale funzione.

Nella figura 12 è riportato il risultato ottenuto utilizzando per votare dei biglietti colorati: in questo caso ogni persona riceve un **numero prefissato** di biglietti che usa come gettoni o *fiches* per votare; la votazione avviene semplicemente attaccando i vari

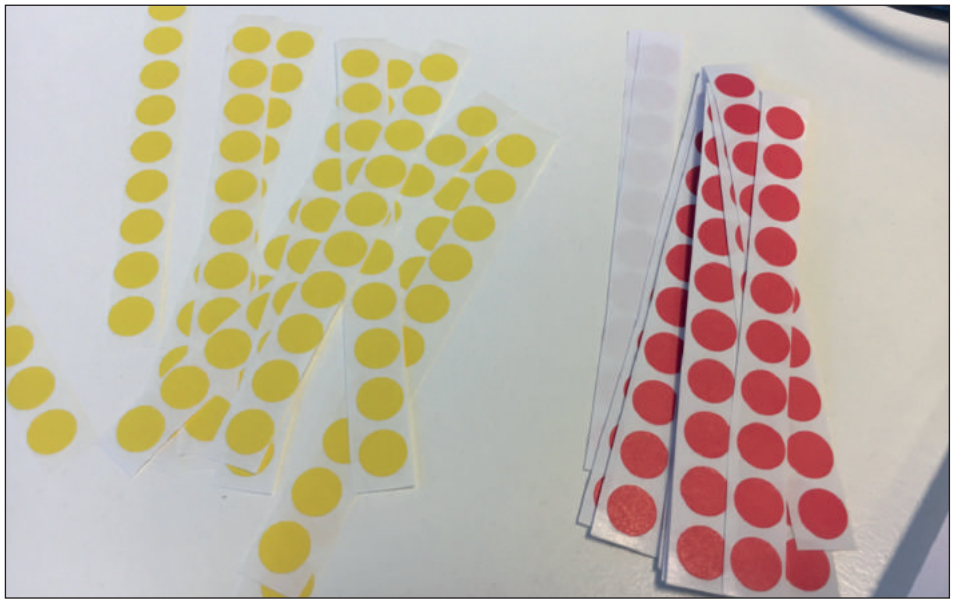


Figura II.11. Con dei semplici pallini adesivi si possono far esprimere le preferenze sulle varie attività.Figura 21 – Il secondo livello intorno al nucleo centrale è quello del "come". In che modo cambierà il comportamento dei vari attori e in che modo possono aiutarci a raggiungere gli obiettivi?

Figura II.12. I vari partecipanti possono votare attaccando i pallini ai cartellini dei viaggi utente.

biglietti ai task che si ritiene siano di maggior valore. Al termine della votazione, analizzando la disposizione dei biglietti o la macchia di colore, si può dedurre, in modo estremamente semplice ma efficace, quali sono i task il cui valore è ritenuto più elevato.

Raramente le varie persone danno gli stessi punteggi per le storie simili: proprio per questo motivo tale attività offre interessanti spunti di **discussione** finalizzata a capire se ci sia **intesa** su cosa si deve fare, su dove si produrrà **valore** e se sia chiaro del perchè si dovranno realizzare le varie funzionalità.

Dopo che il team ha discusso sulle varie differenze sui punteggi, si può quindi provare a unire le varie soluzioni, accorpando, sostituendo, cambiando.

A questo punto il team si muove di fronte alla board camminando da destra a sinistra osservando le **varie attività** e cercando per un'ultima volta di vedere se ci si è dimenticati di qualcosa, se è necessario nuovamente spostare qualche task da una colonna a un'altra. Si possono prendere **appunti** sulla mappa, segnalare i cartellini corrispondenti alle parti più **rischiose** o indeterminate e provare a stimolare la discussione in modo informale sui vari cartellini appesi. Una conferma che il lavoro svolto è utile viene data se nascono

Figura II.13. Tramite il gioco Buy An Issue, si può stimolare in modo estremamente rapido ed efficiente il confronto fra le persone del team. In questo esempio, ogni persona riceve un numero di biglietti colorati prefissato e uguale per tutti. Questi bigliettini servono per votare i task scegliendo quelli che si ritiene abbiano maggior valore. In base alla disposizione dei biglietti (pattern analisys) e alle macchie di colore, si può rapidamente dedurre quali sono i task ritenuti più importanti.

commenti del tipo "mi era nota l'importanza di questa funzione, ma in questo contesto appare chiaro che sia ancora più importante e rischioso non implementarla in modo corretto", oppure "questa cosa non l'avevo minimamente presa in considerazione".

Prioritizzazione dei task

Il passo successivo è quello di introdurre la **dimensione temporale** rispetto alla quale il gruppo implementerà le funzionalità del sistema. In tal caso ha certamente poco senso ordinare le colonne con le **User Activities**, visto che queste sono funzionalità di alto livello delle quali tutte conterranno al loro interno alcune parti più urgenti insieme ad altre che potranno essere realizzate in un secondo momento.

Riprendendo l'esempio portato da Jeff Patton nel suo libro, si potrebbe immaginare un progetto che debba ideare, disegnare e poi costruire un'automobile. In questo caso le varie macrofunzionalità potrebbero essere le **User Activities** "impianto frenante", "sistema di propulsione", "impianto sterzante", "carrozzeria", "struttura portante" etc.

In questo caso è veramente difficile, se non inutile, utilizzare del tempo per capire se sia più importante la activity **impianto frenante** o quella relativa al **motore** o alle **ruote**. Dette in questo modo, sono tutte importanti e non è possibile stabilire in alcun modo a cosa convenga dare priorità: un'eventuale implementazione **minimum viable product** (**MVP**) di automobile che includa solo una o l'altra sarebbe comunque inutile se non addirittura pericolosa.

Priorità dei sottocomponenti

Entrando invece nel dettaglio delle varie **Activity** è evidente come sia possibile optare per una qualche forma di prioritizzazione basata sull'importanza dei **sotto-task**: per esempio, dell'impianto frenante potremmo rimandare a un secondo momento l'implementazione di un sistema di antibloccaggio delle ruote; l'ABS è importante ma può essere considerato un meccanismo da introdurre in un secondo momento, certamente dopo che si sia realizzato il prototipo da far vedere agli utenti finali. Discorso analogo per la carrozzeria o per il motore: ci sono molte parti e funzionalità che potrebbero essere inserite in un secondo momento, dopo la realizzazione del prototipo.

Tornando alla mappa che si sta realizzando, si può quindi introdurre un **asse verticale** sul quale spostare i vari task in base all'urgenza con la quale si vorranno implementare. Questa attività di ordinamento è ovviamente in carico al **Product Owner**, anche se potrà essere aiutato da altre persone; è comunque sua la **responsabilità ultima** dell'**ordine** delle cose da fare.

La prima riga: MVP

A questo punto, osservando la **prima riga** che si forma in alto vicino alla linea della **backbone**, si potrà constatare che essa contiene le storie che possono dar vita a una **versione minimale** dell'applicazione con tutte le **funzionalità essenziali** per gli utenti finali. La prima riga rappresenta quindi la versione **Minimum Viable Product**

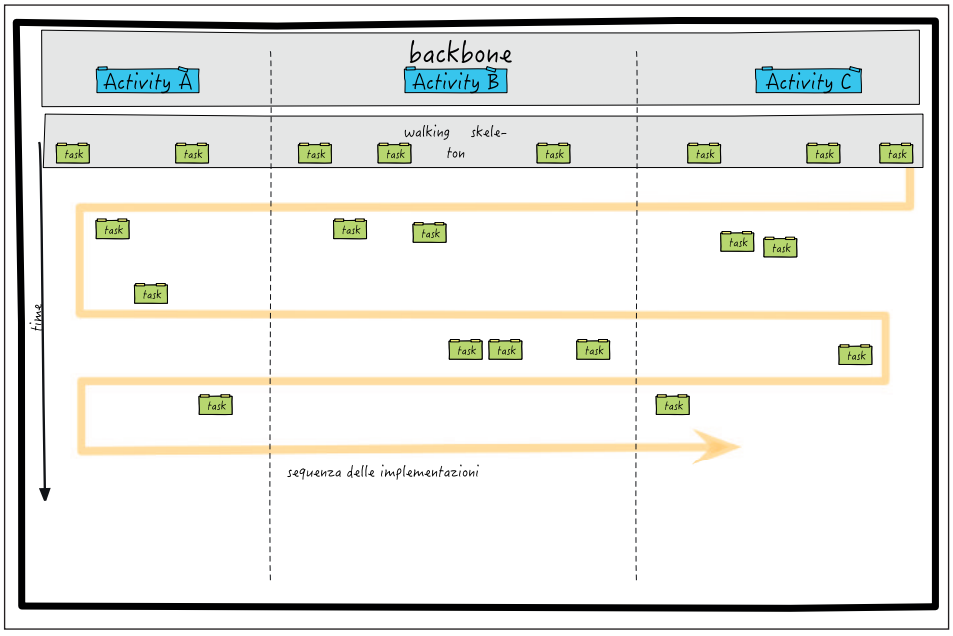


Figura II.14. La sequenza di implementazione delle storie compone un ordinamento verticale con uno orizzontale, e può essere letta con uno schema che ricorda quello della scrittura bustrofedica.

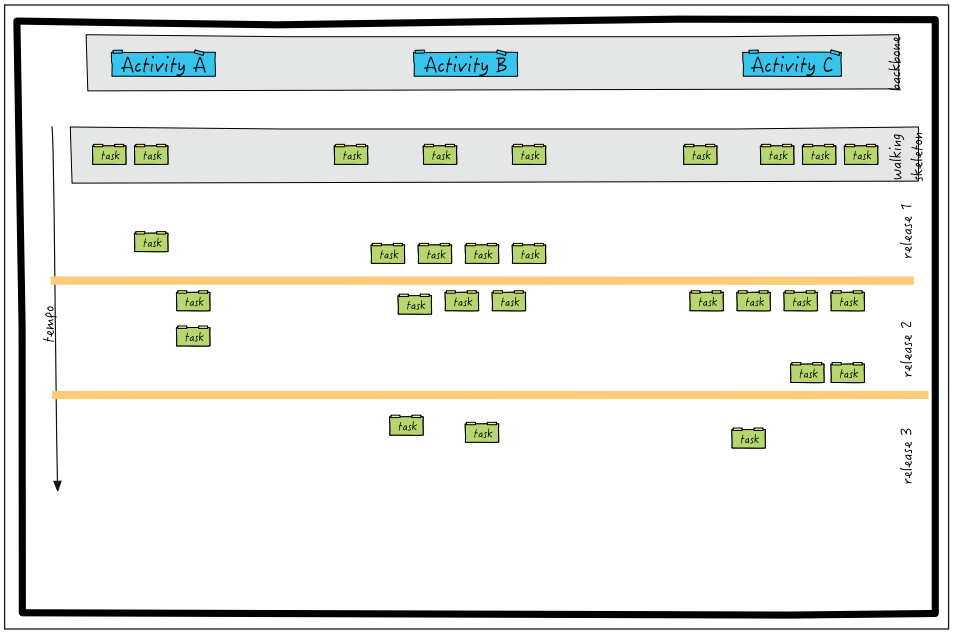


Figura II.15. Prioritizzazione verticale dei task: nasce il walking skeleton. Tracciando delle linee orizzontali si possono creare delle corsie corrispondenti alle varie release.

(**MVP**) del prodotto: prendendo in prestito una definizione di Alistair Cockburn, è detta **Walking Skeleton**, con una immagine molto eloquente di uno scheletro (quindi solo la struttura portante) in grado comunque di camminare (quindi di svolgere alcune funzioni fondamentali). Sul sito di Cockburn si trova questa interessante definizione [5]:

«[il walking skeleton] è una implementazione minimale del sistema in grado di svolgere una piccola funzione end-to-end. Non necessita di usare l'architettura finale, ma dovrebbe collegare insieme i principali componenti architetturali. La parte architetturale e la parte funzionale possono poi evolvere in parallelo».

Release

Dopo aver individuato la priorità circa l'implementazione dei vari task, grazie all'introduzione della dimensione **temporale**, si può raffinare questa suddivisione introducendo il concetto di **release**, disegnando delle strisce in orizzontale (figura 15).

Conclusioni

La **User Story Map** è uno strumento molto utile e versatile che richiede alcune conoscenze e un po' di pratica, ma che risulta molto chiaro e capace di irradiare conoscenza a chi lo utilizzi.

Consente una definizione **collaborativa** dei vari task, una loro **organizzazione** in macroaree funzionali, una prioritizzazione in termini di **valore** delle varie attività e di **tempi** di realizzazione delle varie funzioni.

Insieme agli altri strumenti visuali analizzati nei capitoli precedenti, la User Story Map contribuisce in maniera strutturata ma flessibile ad affrontare il percorso che porta dalla visione al prodotto finale.

Riferimenti

[1] Jeff Patton, *User Story Mapping: Discover the Whole Story, Build the Right Product*. O'Reilly, 2014

[2] Giovanni Puliti, *#Play14. Reportage dalla unconference sul serious game*. MokaByte 194, aprile 2014

[3] Hoang C. Huynh, *Prototipare con MEAN.io. II parte: Le cose importanti prima di tutto*. MokaByte 198, settembre 2014

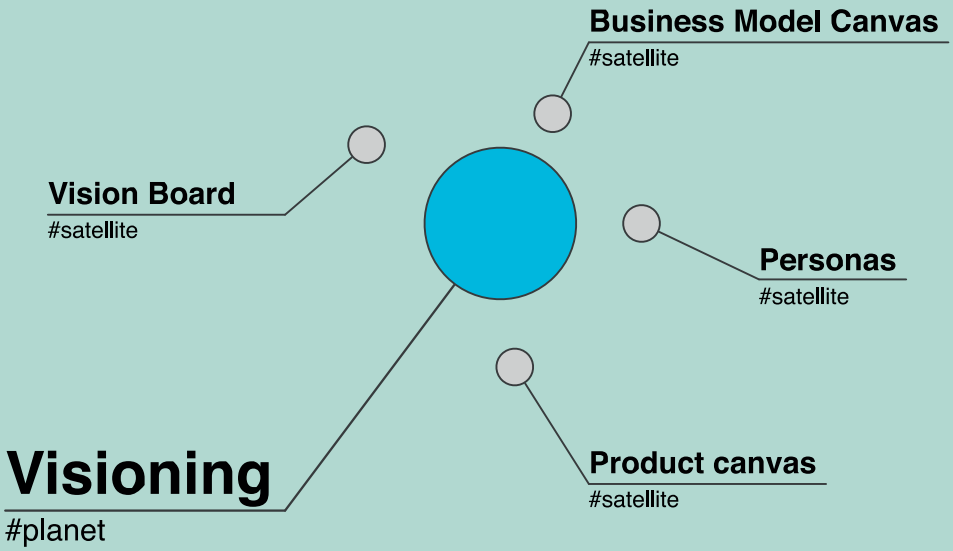
[5] Walking skeleton
http://alistair.cockburn.us/Walking+skeleton

[6] Adzic G. – Evans D. – Korac N., *Fifty Quick Ideas To Improve Your User Stories*. Neuri Consulting LLP, 2014

[7] How to split a user story
http://goo.gl/0k21t0

[8] Silent Brainstorming: A Guide To Using Post-its
http://goo.gl/65WdeZ

Capitolo 5
Raccogliere i requisiti con il Product Canvas



I requisiti con il Product Canvas

Una volta definita e condivisa la **Vision Board**, si può provare a stilare un primo **elenco** delle cose da fare creando una prima versione del **Product Backlog**; per svolgere questo compito si può utilizzare la tecnica vista in precedenza dello **Story Mapping**, oppure sfruttare un altro strumento come il **Product Canvas**. Nella figura 16 è mostrato il tipico impiego della **Product Canvas**, che si inserisce all'interno di un processo **iterativo** e **incrementale** e che consente di trasformare un'idea in un elenco di **funzionalità**, in questo caso **storie utente**, da implementare nella fase di lavorazione vera e propria.

Per comprendere meglio l'utilizzo di questo strumento, conviene entrare nel dettaglio di come è fatta la "lavagna". La figura 17 riprende un modello proposto da Roman Pichler [2] e ci mostra le varie sezioni della Product Canvas.

La **Product Canvas** normalmente viene "compilata" in un **workshop** in cui il team arriva a completare le varie parti della board in forma di prima realizzazione. In genere, la durata di tale evento è quello di una giornata completa.

Vediamo di seguito le diverse sezioni e le informazioni che andranno inserite.

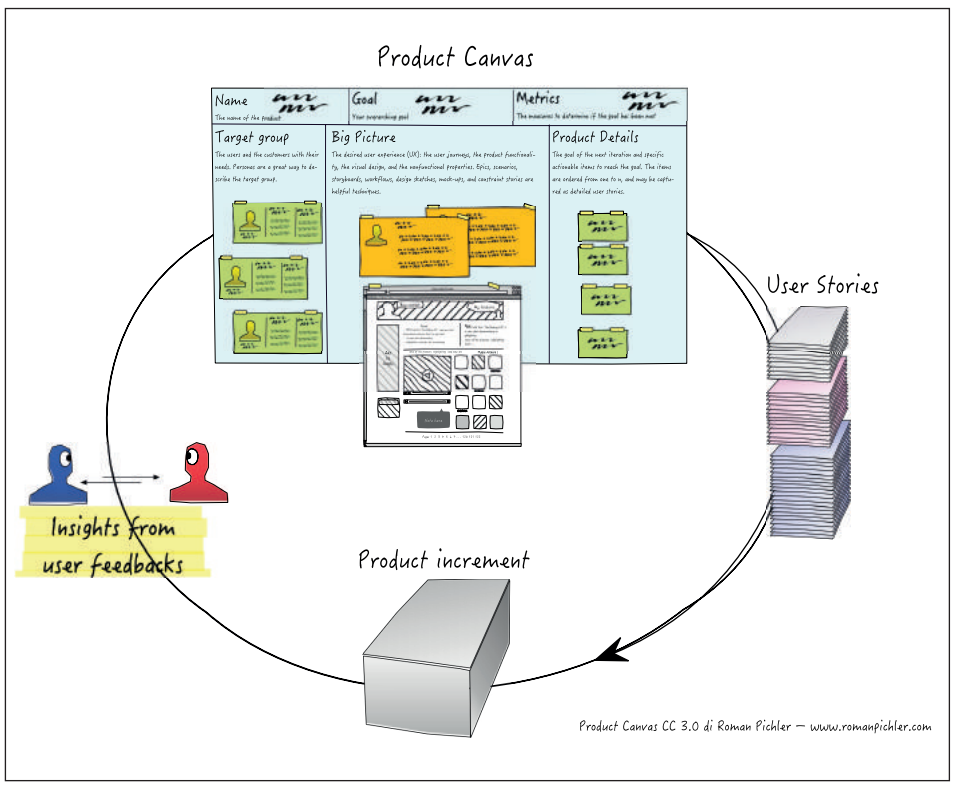


Figura II.16. Lo schema di lavoro iterativo e incrementale all'interno del quale si inserisce l'uso della Product Canvas.

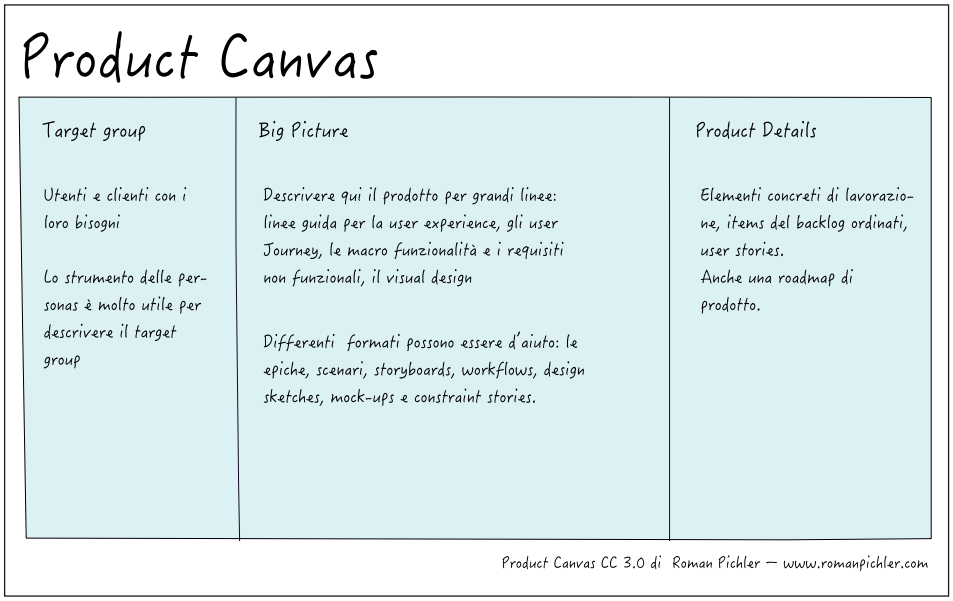


Figura II.17. La Product Canvas con le sue sezioni.

Nome

Il **nome** del prodotto che si deve realizzare. Specificare anche la versione del prodotto, quando è un redesign o la nuova implementazione di qualcosa che già esisteva.

Goal

Il **goal** rappresenta l'obiettivo che si vuole raggiungere con la realizzazione di questo prodotto o tramite questa nuova release di un prodotto già esistente: trovare nuovi utenti, soddisfare nuove richieste di mercato e così via. Roman Pichler, nel suo sito, propone l'uso della **Go Product Roadmap** [3]come strumento di pianificazione del prodotto orientata alla definizione degli obiettivi. Nel caso in cui si sia scelto di utilizzare questo strumento, per approfondire l'analisi del tema "obiettivi", si potranno riportare nella Product Canvas le parti corrispondenti.

Metriche

La sezione **metriche** serve per specificare dei parametri di valutazione che possano permettere di capire quanto quello che si sta realizzando rispetta gli obiettivi prefissati. Anche in questo caso la Go Product Roadmap può essere di aiuto.

Target Group

La definizione del **gruppo di utenti tipo** serve per individuare chi potrebbero essere gli utilizzatori tipici di questo prodotto, e capire successivamente i loro bisogni. Per

compilare questa parte si può usare il formalismo delle **personas** e per questo si potrebbero utilizzare le personas individuate durante la fase di visioning e che potrebbero essere qui riutilizzate in forma sintetica prendendo direttamente spunto da quanto inserito nella **Vision Canvas**. La Product Canvas offre infatti una visione di sintesi da un differente punto di vista, per cui non deve meravigliare che si ricavino in questa fase informazioni differenti a complemento di quanto visto durante la vision.

Visto che nella **Product Canvas** il focus è sulle funzionalità del sistema, potrebbe essere utile in questo momento provare a **ordinare** le **personas** in **funzione** dell'**importanza** dell'utente: quello che paga o uno sponsor importante, o un'altra categoria importante. Si può anche ordinare le personas in funzione dell'importanza dei bisogni: urgenza, funzionalità "rivendibilità", ROI maggiore o altro. In questo modo sarà certamente più semplice poi stabilire la priorità delle funzionalità da implementare. Alle **Primary Personas** così individuate dovrebbero infatti corrispondere funzionalità che poi saranno in cima al backlog di prodotto.

Big Picture

La **Big Picture**, vale a dire il "quadro d'insieme", descrive cosa è necessario fare per soddisfare i bisogni delle personas: dovrebbe fornire la visione di insieme del prodotto ad alto livello; concettualmente ricorda l'**indice di un libro**: ne descrive il contenuto senza entrare nel dettaglio.

Spesso in questa parte si inseriscono gli **User Journeys** ovvero la descrizione delle varie operazioni che l'utente svolge quando utilizza il prodotto: si collega, apre il menu, cerca e sceglie, compra, stampa, esce etc. Per la definizione dei "viaggi" possono usare tecniche come gli **Scenari** [4] o lo **User Story Mapping** che abbiamo visto al Capitolo 4.

Durante questa fase, è utile anche la raccolta di informazioni relativamente alla parte non funzionale, ai requisiti tecnici e alle cosiddette **constraint stories** che possono in qualche modo impattare sulla esperienza utente o sulla architettura software.

Per comprendere meglio l'esperienza utente spesso si realizzano in questa fase delle immagini di interfaccia utente, che poi si "attaccano" direttamente alla board. Per questo lavoro si possono usare diverse tecniche come **design sketches** e **mock–up** [5], oppure applicazioni per la prototipazione rapida delle interfacce che permettano di provare una versione semi funzionante in tempi rapidissimi [6]. In questa fase, è importante concentrarsi sugli aspetti strutturali del design grafico (layout di massima) e sugli aspetti critici legati all'architettura dell'informazione.

Product Details

Infine, nella sezione **Product Details**, si inseriscono gli elementi che dovranno essere messi in lavorazione nella prossima iterazione (lo sprint zero se siamo a inizio progetto).

Questo elenco può essere prodotto analizzando i risultati del punto precedente. Per esempio i vari passi dei diversi "viaggi utente" possono essere visti come gli elementi a

grana grossa (**epiche** o **features**) del **Product Backlog**. Se si è utilizzata la tecnica dello **User Story Mapping**, il set delle storie individuate per il prossimo sprint potrebbe essere inserito così com'è in questa parte.

Molto utile usare la Product Canvas in congiunzione con la tecnica dello User Story Mapping perché consente di produrre contenuti interessanti tramite un lavoro strutturato e rigoroso. Nella figura 18, infine è riportato un esempio di una Product Canvas per un progetto mobile.

L'utente prima di tutto

La canvas è progettata per permettere che il processo relativo alla generazione delle varie parti si traduca in un **flusso** di informazione da sinistra a destra, ossia dalle personas ai dettagli del prodotto. In questo, modo si pone sempre l'**utente** al **centro** del proprio lavoro concentrando quindi gli sforzi sui suoi bisogni e sui suoi desideri, permettendo la realizzazione di un prodotto che li possa soddisfare.

Il flusso di informazioni

Interessante notare come questo **flusso** di **informazioni** non si realizzi solamente all'interno della board, ma sia lo stesso tipo di percorso che si ritrova più in grande nel processo di lavorazione (figura 16): in quel caso, grazie al processo iterativo e incrementale messo in atto, ogni incremento nella realizzazione del prodotto finito permette

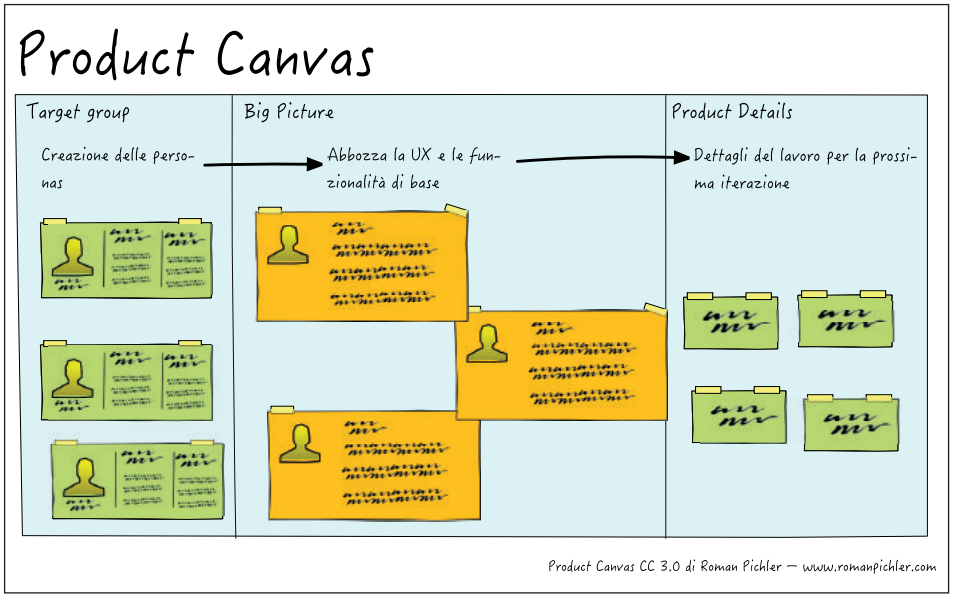
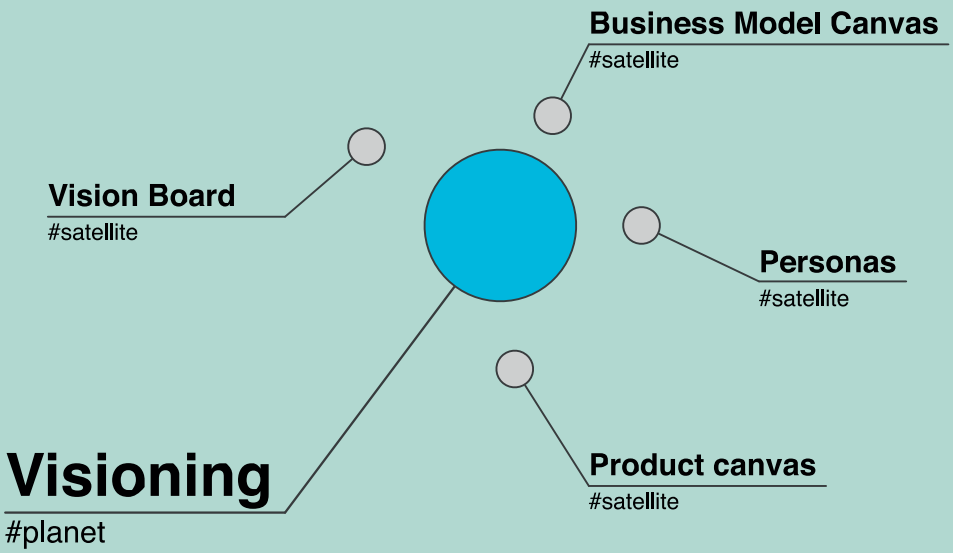


Figura II.18. Il processo di produzione delle informazioni segue un flusso da sinistra a destra ed è incentrato sul concetto di utente del prodotto.

di ricavare informazioni direttamente dagli utenti finali e quindi di migliorare la conoscenza del dominio, delle esigenze da soddisfare, ossia, in definitiva di quello che si deve fare. Si realizza quindi un circolo virtuoso in cui la Product Canvas può essere vista come una sorta di **strumento di apprendimento** utile sia per abbozzare le idee iniziali che per raffinarle in un secondo momento.

Per questo motivo non deve stupire se la Product Canvas subisce un processo di **correzione** e **modifica** per inserire nuovi dettagli, nuove informazioni o per cambiare quanto già inserito, man mano che si scoprono nuove cose. Per questo spesso le board sono piene di foglietti, scritte, scarabocchi: sono la brutta copia del tema in classe, dove continuamente si fanno correzioni, si aggiungono parti e se ne cancellano altre.

Capitolo 6
Organizzare i deliverables di progetto tramite le Impact Maps



Un'alternativa intuitiva

Per arrivare a una organizzazione delle storie utente e al relativo raggruppamento per funzionalità e necessità utente, un utile strumento, da usare in alternativa ai vari story mapping e product board, è la **impact map**, introdotta da Gojko Adzic [1]: si tratta di un tool dall'approccio estremamente intuitivo — si ispira alle **mappe concettuali** o *mind maps* — che connette in maniera visuale i vari *deliverable* di prodotto e gli obiettivi di business.

Come funziona una impact map

Una **impact map** è di fatto una mappa mentale (**mind map**) che offre una visualizzazione dello *scope* di un prodotto e mette in relazione le varie ipotesi connesse per la sua realizzazione.

La sua realizzazione avviene in modo collaborativo e incrementale, durante la discussione che viene condotta e in un certo senso facilitata rispondendo a 4 semplici domande:

- **Perché**? Per quale ragione dovremmo realizzare questo prodotto? Qual è lo scopo? Quali sono gli obiettivi che ci poniamo? Quale la vision?
- **Chi**? Quali sono gli attori che potrebbero abilitare/influenzare il raggiungimento dell'obiettivo?
- **Come**? In quale modo potrebbe cambiare il comportamento degli attori una volta realizzato il prodotto? Ma anche, come team di sviluppo in che modo possiamo abilitare questo cambiamento?
- **Cosa**? Che cosa potremmo realizzare, sempre dal punto di vista del team di sviluppo.

Perché: definire l'obiettivo

Rappresenta il nodo centrale della mappa e permette di rispondere alla domanda forse più importante: **perché** dovremmo fare questo prodotto/servizio/iniziativa?

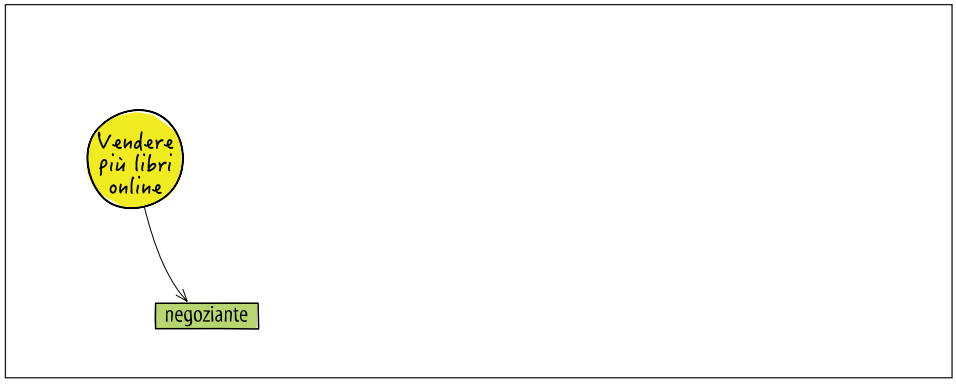


Figura II.19. Il nodo centrale della Impact Map è rappresentato dalla risposta al "perché" si vuol realizzare il nostro prodotto.

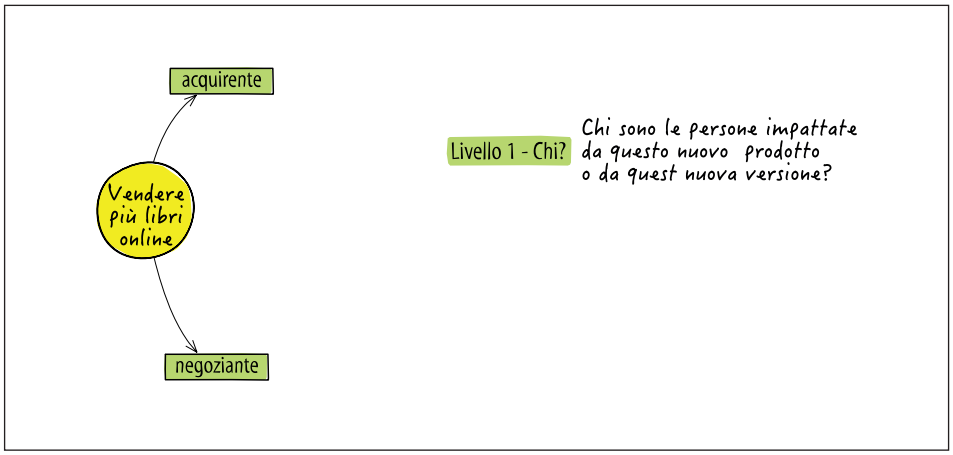


Figura II.20. Intorno al nucleo centrale, sviluppiamo un primo livello con le risposte alla nostra domanda "chi?", riguardante sia chi è il destinatario impattato dal prodotto che vogliamo sviluppare, sia chi può produrre l'effetto desiderato.

Rispondendo a questa domanda si riesce a descrivere l'obiettivo che si intende raggiungere. Una **impact map** è pensata per descrivere un prodotto nella sua **interezza**: nel nodo centrare della mappa troviamo quella che forse è la parte più importante e che permette spesso di descrivere in modo sintetico la vision di prodotto, o meglio **una parte** della **vision** (obiettivi, scope).

Il focus dovrebbe sempre essere sull'individuare ed evidenziare le **necessità** degli utenti, più che sul fornire soluzioni: pensare alle necessità che si vogliono soddisfare, invece che al modo in cui fornire un supporto. Per raggiungere questo obiettivo si potrebbe utilizzare una frase, uno slogan, o anche qualcosa di più strutturato come un **elevator pitch**.

Spesso non è detto che la visione sia completa a inizio progetto: per questo può valere la pena iniziare a compilare la mappa inserendo una versione abbozzata della visione per poi raffinarla e rivederla man mano che si chiariscono meglio gli obiettivi tramite l'analisi e la compilazione degli altri livelli della mappa.

Scope e **vision** di un progetto dovrebbero essere sempre ben chiari a tutti i membri del team, e non solo di questo. Pertanto spesso è utile rendere la mappa — ma soprattutto il suo nodo centrale — accessibile e visibile, per esempio appendendola a una parete.

È infatti estremamente importante che tutti sappiano cosa fare e come muoversi nel caso si debbano prendere decisioni in modo rapido, intervenire in emergenza, apportare correzioni al progetto, gestire cambi di rotta o tamponare emergenze.

Chi: per chi stiamo sviluppando il prodotto

Questo livello della mappa serve per rispondere alla seguente domanda: **chi** può **produrre** l'effetto desiderato? Chi potrebbe impedirlo? **Chi** sono gli **utenti**/**destinatari** a

cui ci vogliamo rivolgere? **Chi** ne sarà **impattato**: l'uso di questa parola non è casuale, essendo alla base del concetto di **impact map**.

A differenza di altri strumenti, in questo caso si prendono in considerazione non solo gli utenti finali, ma anche quelle persone/realtà che possono **influenzare** la realizzazione del prodotto con le loro decisioni.

Un modo per definire il concetto di qualità — molto usato all'interno della comunità agile — è "valore rilasciato all'utente finale" che quindi deve essere ben compreso e descritto; devono essere comprese le sue necessità, i suoi obiettivi e le sue preferenze per indirizzare meglio le varie componenti o declinazioni del prodotto finale. Di fatto la maggior parte dei tool o tecniche di Product Definition insistono molto su questo aspetto: si pensi alle **Personas**.

Nel processo di creazione di una Impact Map la definizione degli attori può essere utile per comprendere le loro necessità e quindi riuscire a prioritizzare meglio le cose da fare.

Un modo per qualificare gli attori potrebbe essere per esempio di suddividerli in base al coinvolgimento con il prodotto che si deve realizzare. Alistair Cockburn suggerisce questa semplice classifica:

- **Attori primari** i cui obiettivi devono essere soddisfatti pienamente.
- **Attori secondari** che non sono coinvolti direttamente ma possono fornire un servizio a supporto del prodotto che si deve realizzare. Sono coinvolti o comunque possono essere un valido supporto.

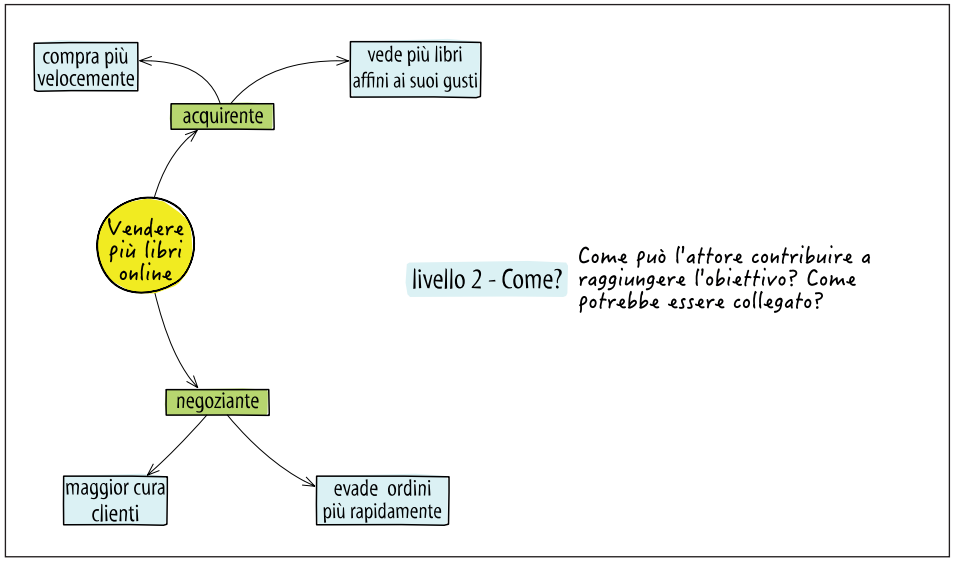


Figura II.21. Il secondo livello intorno al nucleo centrale è quello del "come". In che modo cambierà il comportamento dei vari attori e in che modo possono aiutarci a raggiungere gli obiettivi?

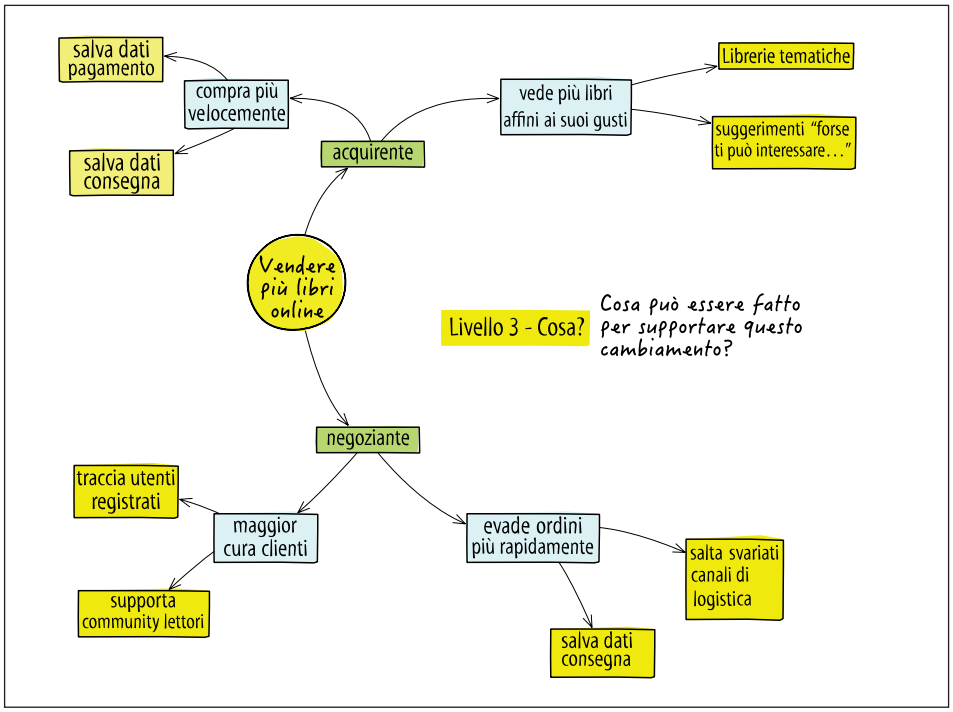


Figura II.22. Con l'aggiunta del terzo livello, quello del "cosa", si arriva a individuare elementi che rappresentano i deliverables, le caratteristiche del prodotto ma anche attività relative all'organizzazione.

- **Off stage actors**, i quali sono interessati di riflesso dal sistema ma non lo usano direttamente o non forniscono un servizio.

Come: in che modo realizzare il prodotto

Questo livello suggerisce il "**come**" si potrebbe realizzare il prodotto e risponde alla domanda: "In che modo dovrebbe cambiare il comportamento degli attori? Cosa ci aspettiamo che succeda? Come possono concretamente aiutarci ad ottenere gli obiettivi? Come possono invece impedire od ostacolarci?".

Questo livello della mappa quindi ci aiuta a capire gli obiettivi degli utenti finali, quali sono le cose che vogliono ottenere; ci permette quindi di descrivere il cosiddetto **change behaviour** dell'utente e quindi quali dovranno essere gli **impatti** del lavoro che si sta realizzando.

Spesso un impatto evidenzia un **cambiamento di comportamento**: per esempio potrebbe essere una buona descrizione "vendere libri online in modo più semplice e rapido", mentre genericamente "vendere libri online" potrebbe essere troppo generico e non introduce nessuna variazione: ci sono già molti e-commerce che lo fanno.

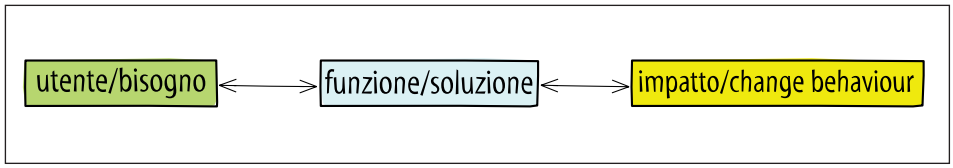


Figura II.23. Nella stesura dei requisiti è necessario concentrarsi non su un solo aspetto, ma collegarli nella "terna" rappresentata da "utente / bisogno", "funzione / soluzione" e "impatto / change business behaviour".

Interessante notare come le differenze fra gli attori evidenziate durante la compilazione del livello precedente (quello del chi), in questo caso si potranno tradurre in impatti che potrebbero essere legati da un legame di priorità, o potrebbero essere complementari o del tutto incompatibili fra loro.

Per non rischiare di divagare troppo, si dovrebbe cercare di focalizzare l'attenzione solo su quegli impatti che possono essere utili a muovere il progetto nella direzione giusta. In pratica, un trucco potrebbe essere di rimanere sulle attività di business da implementare, non genericamente divagando su tutte le idee che ci passano per la testa; altra indicazione fondamentale è di non cadere nel tranello di elencare un set di funzionalità del prodotto finale.

Cosa: cosa facciamo per portare gli impatti

Questa sezione della mappa permette di rispondere alla domanda: "Come team di sviluppo/azienda/organizzazione, **cosa possiamo fare** per supportare o implementare il gli impatti desiderati? Ossia cosa possiamo fare per abilitare il cambio di comportamento?".

Gli elementi qui presenti corrispondono ai cosiddetti *deliverables*, le funzionalità del software, oppure semplici attività organizzative.

Dovrebbe ormai essere chiaro quanto sia importante collegare i deliverables di progetto con i corrispondenti obiettivi di business e per ogni collegamento argomentarne le implicazioni (ossia gli impatti) di business: detto in altro modo è necessario definire la terna illustrata nella figura 23.

Collegando i deliverables con gli obiettivi e le conseguenze, è come se la mappa ci aiutasse a definire il filo logico che collega una scelta di business con le eventuali soluzioni che possono risolverlo. L'**obiettivo** di una **Impact Map** è esattamente questo, ossia creare un **collegamento** fra **quello** che deve essere fatto, **perché** deve essere fatto e **per chi**.

Grazie alla struttura organizzativa a grafo, una impact map è di grande aiuto per organizzare gli eventuali rilasci intermedi: i rami (suddivisione verticale) o i livelli (suddivisione orizzontale) potrebbero essere associati alle milestone o release. Anche le eventuali priorità che si instaurano fra gli attori sono di grosso aiuto per definire la sequenza delle cose da realizzare.

Questo livello della mappa è il più importante e per questo dovrebbe essere completato in modo incrementale e iterativo.

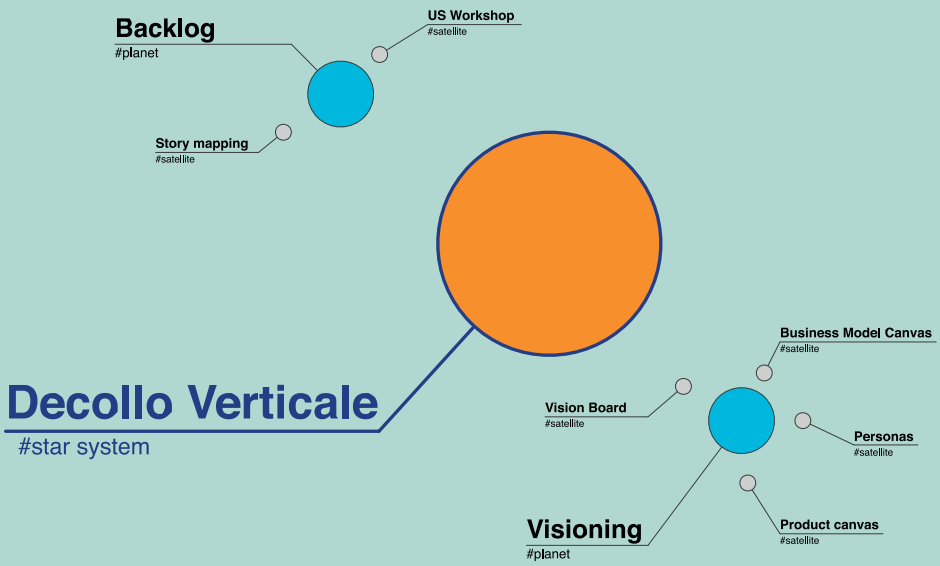
Man mano che si rilasciano nuove parti, si potrebbero aggiornare le parti ancora poco complete. I deliverables sono opzioni, non tutto quello che verrà qui rappresentato potrebbe essere implementato.

Non conviene fissarsi fin da subito sui dettagli, ma è preferibile partire dalla definizione di alto livello dei deliverables; successivamente si potranno raffinare tramite le consuete tecniche di splitting come si fa comunemente sugli elementi di un backlog.

Riferimenti

[1] Gojko Adzic, *Impact Mapping: Making a big impact with software products and projects*. Provoking Thoughts, 2012

Capitolo 7
Mettiamo tutto insieme



Strumenti, ma non solo...

Fin qui, abbiamo illustrato **strumenti** per svolgere uno dei compiti più difficili: **trasformare** un'**idea** in un **prodotto**. E se li abbiamo visti all'opera nell'arte di sviluppare software, non va dimenticato che molto di quanto raccontato in questa Parte 2 del libro è applicabile anche in altri contesti produttivi. Si tratta di un lavoro che comprende la definizione dell'**idea** di **base**, la **formalizzazione** dei **requisiti** e l'**impostazione** del **progetto**.

La **condivisione** e la **comprensione comune** dell'idea sono forse l'ostacolo più grande per far partire un progetto. Si possono trascorrere giorni e giorni in interminabili riunioni per condividere l'idea a tutti i livelli: possiamo coinvolgere ruoli direttivi e finanziari, project manager, sviluppatori, marketing, ufficio vendite e supporto clienti... per poi scoprire di avere una idea che è solo **superficialmente condivisa** e che il consenso sul progetto è dato più sulla fiducia che sul contenuto. Spesso la **comunicazione verbale** non aiuta più di tanto, perché molti aspetti dell'idea del prodotto in molti casi **non** sono facilmente condivisibili a parole.

Vantaggi e criticità della documentazione scritta

Trasmettere le informazioni **per iscritto** è una prassi molto comune che ha indubbi vantaggi:

- **formalizza** il pensiero e la comunicazione verbale, **organizzando** in modo logico i **contenuti**;
- consente di mantenere uno **storico** delle scelte adottate, e magari anche delle ragioni alla base di tali scelte;
- il documento **scritto** ben si presta a inserirsi in un **processo** di **approvazione**.

A fronte di questi vantaggi, quando si decide di scrivere un **documento testuale**, indipendentemente dal formalismo utilizzato, spesso si corre il rischio di concentrare l'attenzione sul "formato" e sulla compilazione del documento piuttosto che sul produrre **valore** per il cliente.

Il secondo punto dell'Agile Manifesto ("Il software funzionante più che la documentazione esaustiva") [1] diventa sia un faro guida sia un punto di discussione forte. Come faccio a **non** scrivere documentazione? In Agile si scrive **documentazione**? Quanta? Quale?

Prima di rispondere a queste domande, conviene anzitutto capire quale sia l'**utilità** che risiede nella documentazione dal punto di vista del cliente finale.

Il valore del documento

In tal senso la documentazione prodotta, soprattutto quella realizzata nelle fasi iniziali di progetto, ha valore se è **condivisa** e **compresa** da tutte le persone coinvolte: lo scopo è quello di permettere di condividere l'idea del prodotto o servizio che si vuole realizzare.

Come già più volte detto, solo la **condivisione** della **visione** di prodotto, degli **obiettivi** e delle **modalità** per raggiungerli, consente di realizzare un prodotto che

soddisfi i bisogni di business dell'utente, cosa che in definitiva è il vero scopo di un progetto software.

Quindi, lo **scopo della documentazione** è **condividere conoscenza** e permettere la collaborazione. Guardando le cose da questo punto di vista, l'obiettivo di produrre documentazione è certamente nobile e utile. Quello che ci si potrebbe chiedere è se ci possano essere **strumenti più efficaci rispetto** alla scrittura di **documenti testuali** che poi spesso non catturano la curiosità degli interessati e finiscono per rimanere in qualche remota cartella.

Canvas come documentazione

Probabilmente l'uso di **lavagne** per disegnare ma anche e soprattutto per **parlare insieme** potrebbe essere una valida alternativa. In questa parte del libro abbiamo già abbondantemente affrontato questo tema, proponendo una serie di strumenti utili per la fase di definizione dei requisiti nonché per la successiva validazione.

Abbiamo parlato di come **identificare** e **formalizzare** la visione di prodotto (**Vision Board**) e di come **individuare** e validare successivamente il **modello** di **business** (**Lean Canvas**); abbiamo affrontato il tema della **traduzione** dei bisogni utente in un **elenco** di **funzionalità** da implementare (**User Story Mapping e Product Canvas**).

In questo ultimo capitolo della Parte II, ricapitoliamo il flusso della lavorazione nel suo complesso, ossia come i vari tool visti fin qui possano essere utilizzati per passare dall'idea al prodotto.

Un riassunto sulle canvas: "from vision to backlog... and back"

Il lavoro di costruzione di un prodotto complesso è frutto di un processo iterativo e incrementale in cui continuamente si parte dalle premesse per realizzare una parte del prodotto finito e, tramite la prova sul campo delle funzionalità inserite, si prova a rivedere, aggiungere e modificare le premesse.

La frase "from vision to backlog... and back" rappresenta questo approccio: la parte del **back**, ossia l'approccio **iterativo** e **incrementale**, permette di trasformare un processo tipicamente **sequenziale** — dalla vision al modello di business alla definizione dell'elenco delle cose da fare — in un processo **iterativo incrementale** tipicamente agile.

Di fatto senza il cortocircuito che, dalla fase di realizzazione, ci fa ritornare ciclicamente a quella di vision e raccolta dei requisiti, quanto visto finora non sarebbe altro che una forma rivista e corretta del caro vecchio waterfall.

Le varie canvas in azione, ossia un approccio iterativo, incrementale e complementare.

Come riportato nella figura 1 al Capitolo 1 di questa Parte II, per passare **dall'idea al prodotto**, la prima cosa da fare è lavorare sulla **visione** per esempio tramite il workshop con la **Vision Canvas**: lo scopo è capire perché si vuole fare questo prodotto, per chi è il prodotto, quali bisogni si vogliono risolvere etc.

Lo scopo in questa fase, lo ricordiamo, è capire in che modo il nostro prodotto "funzioni" in termini di sostenibilità economica e possibilità di guadagno, e come gli utenti saranno interessati a continuare a utilizzarlo, magari pagando per certe funzioni, è compito della **Business Model Canvas**, che si occupa, appunto del **modello di business**.

Quando infine si arriva a entrare nello specifico delle **funzionalità** del prodotto, la **Product Canvas** può rappresentare un interessante punto di congiunzione fra le due attività che si sono svolte parallelamente in precedenza.

Da un lato la **Product Canvas** permette di approfondire in modo efficace un aspetto solamente accennato nella **Business Model Canvas**, ossia quello della **Value Proposition**; dall'altro la parte di raccolta delle varie **personas** offre una visione di sintesi di quello che nella **Vision Canvas** è invece analizzato nell'elenco degli utenti e dei bisogni.

Tre strumenti, un solo processo

Vision Canvas, **Business Model Canvas** e **Product Canvas** sono quindi tre strumenti che offrono una visione **complementare** del sistema che si deve realizzare. Tipicamente, la corretta maniera di usarli è in modo continuativo, iterativo e incrementale:

- in genere si parte dall'impostazione della **vision**;
- si passa subito dopo alla definizione del **modello** di **business**;
- appena si hanno sufficienti informazioni, si può iniziare a impostare lo studio del **prodotto** tramite la Product Canvas;
- da questa, direttamente o tramite l'ausilio dello **User Story Mapping**, si può ricavare una prima versione del **Product Backlog Items**;
- infine, grazie a una metodologia iterativa e incrementale come Scrum, si potrà arrivare alla realizzazione di una prima versione del prodotto che implementi in forma minimale una prima risposta ai bisogni degli utenti individuati (il cosiddetto MVP);
- a questo punto, grazie alla raccolta del feedback degli utenti che hanno modo di provare una prima release di prodotto, si potrà procedere alla **revisione** o al **completamento** delle varie **canvas**.

Se il progetto è gestito tramite Scrum, ad esempio, le informazioni raccolte durante le sprint demo potranno fornire spunti per **completare** o **modificare** sia la parte di **vision**, sia quella di **business**. E, molto probabilmente forniranno dati interessanti anche per completare la raccolta delle **funzionalità** da implementare o per apportare modifiche alla modalità di interazione dell'utente con il sistema (i cosiddetti **user journeys**).

Questo flusso di lavorazione iterativo e incrementale, in cui i diversi strumenti sono messi in relazione complementare, sono mostrati in figura 24. L'idea si condivide e si elabora con l'aiuto della Vision Canvas, e una volta che la vision è chiara e condivisa si possono creare in parallelo Business Model Canvas [7] e Product Canvas; a partire da quest'ultimo, poi si può creare il Product Backlog del prodotto. E poi, con il feedback degli utenti, è possibile apportare le necessarie modifiche ai vari strumenti.

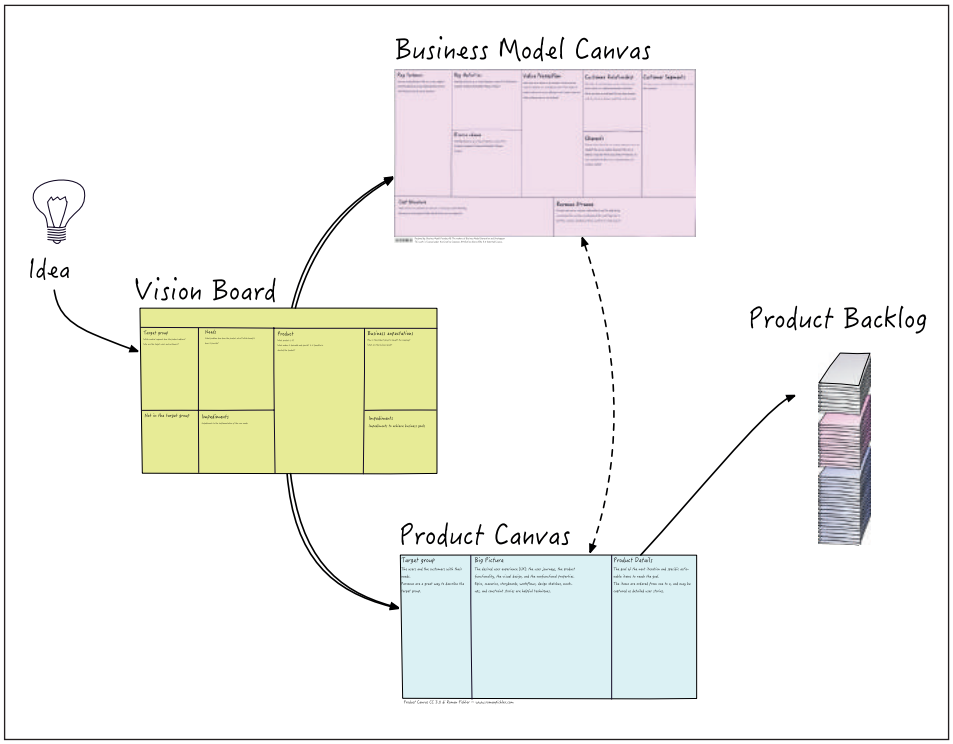


Figura II.24. Visione di insieme degli strumenti: il processo è iterativo e incrementale e si usano tutti gli strumenti per facilitare il percorso da Idea a backlog.

Conclusione

Termina qui la parte dedicata al tema "Dall'idea al prodotto". Questo argomento spesso viene trattato con titoli differenti, come **Lift Off**, **From Vision to Backlog**, **Inception** e altri ancora, ma i concetti e gli strumenti sono quelli trattati in questi capitoli.

Riferimenti

[1] Manifesto per lo sviluppo agile di software
http://agilemanifesto.org/iso/it/

[2] Pichler Consulting
http://www.romanpichler.com

[3] La GO Product Roadmap
http://www.romanpichler.com/tools/product-roadmap/

[4] Gli Scenari secondo Pichler
http://www.romanpichler.com/blog/agile-scenarios-and-storyboards/

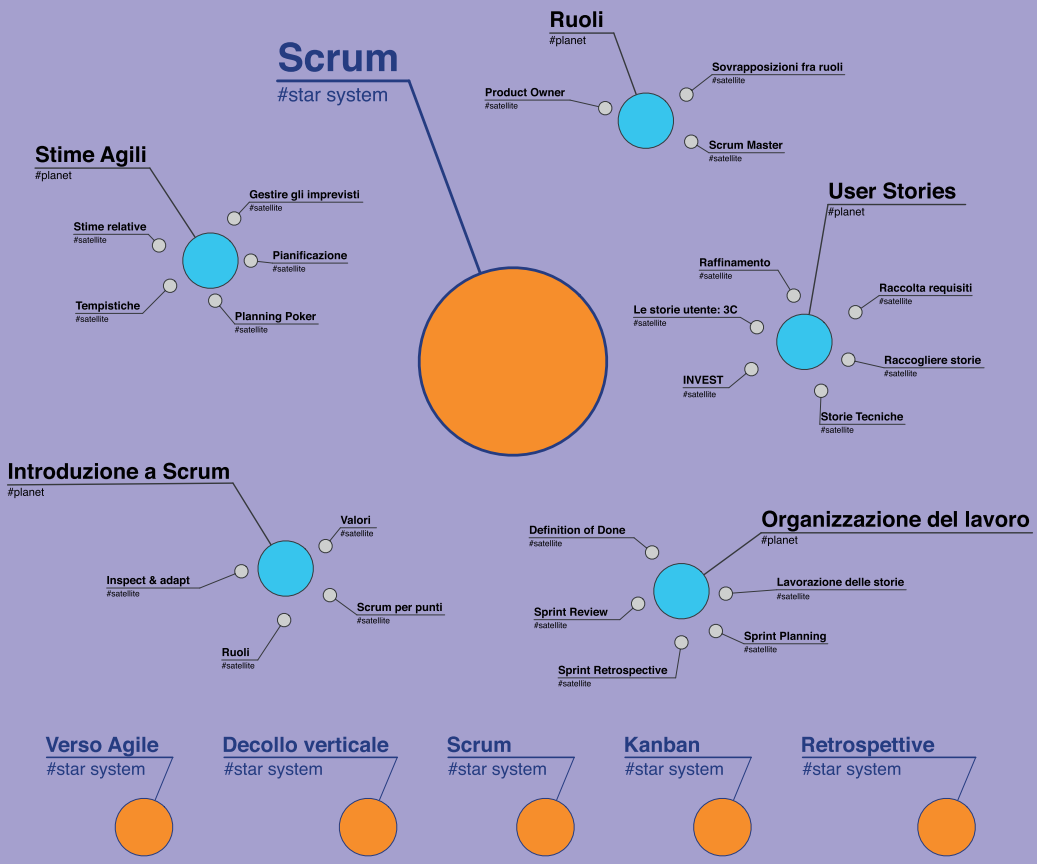
[5] Alcuni strumenti per il design agile di interfacce utente
http://www.romanpichler.com/blog/agile-user-interface-design/

[6] Prototyping On Paper, un'app per la prototipazione rapida di app mobile
https://popapp.in

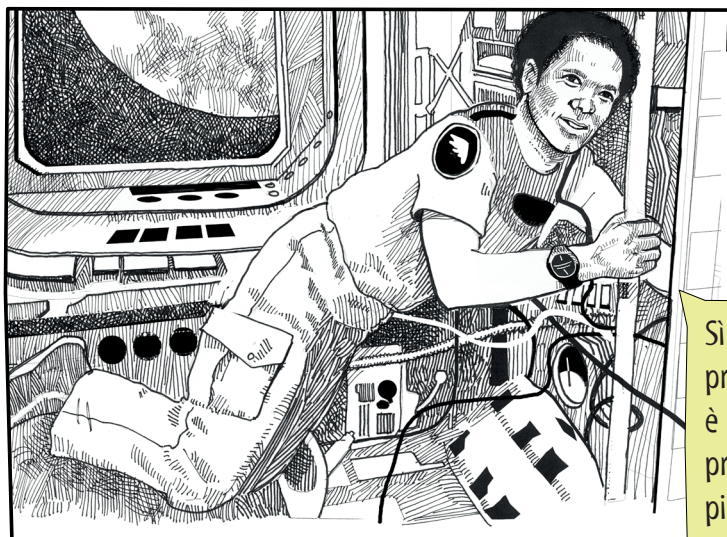
[7] Value Proposition Design
http://www.businessmodelgeneration.com/

PARTE III
IL FRAMEWORK SCRUM

Giovanni Puliti



Parte 3
Il sistema Scrum

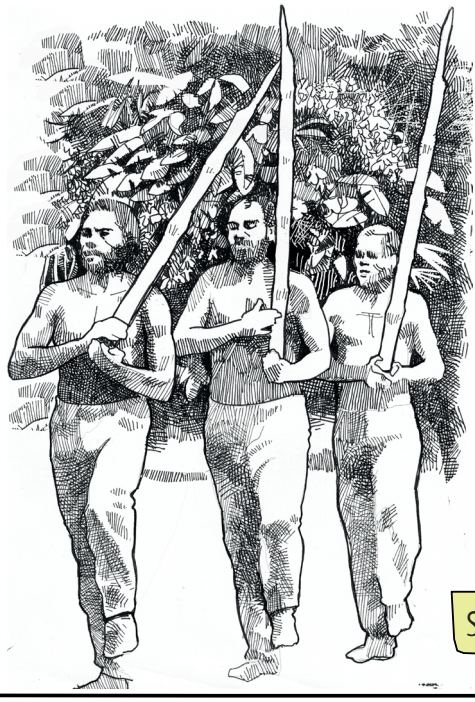


…Ramonek siamo arrivati?

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!

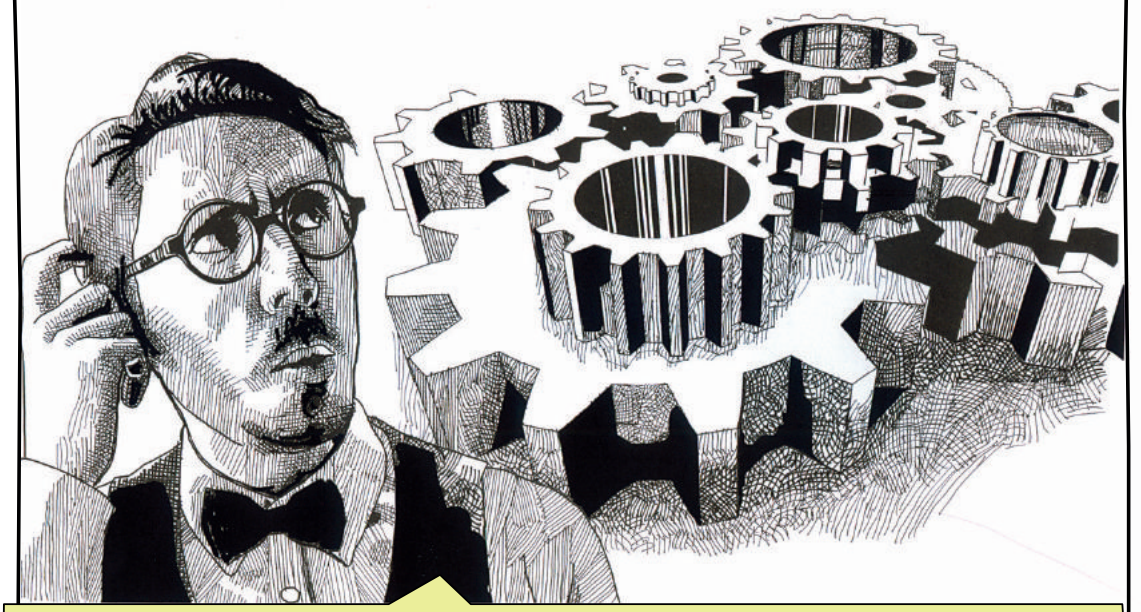
Ora che abbiamo visto cosa ha portato quel gruppo di agilisti a scrivere l'Agile Manifesto in quel famoso fine settimana sulla neve, possiamo passare a comprendere i dettagli tecnici delle metodologie agili, Scrum e Kanban.

E cosa vedremo su questo pianeta?

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

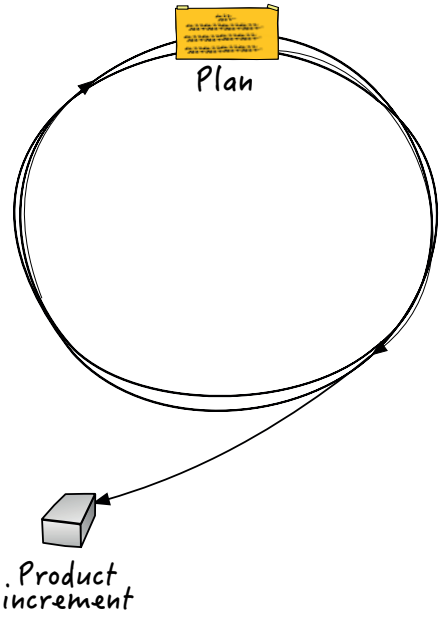
Scrum
star system

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.

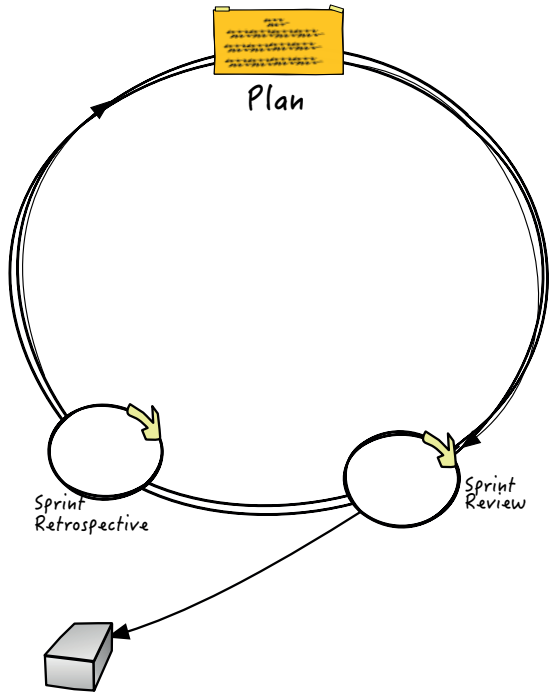


Per questo si lavora prima di tutto in modo da pianificare ogni iterazione tramite una riunione che si chiama Sprint Planning.

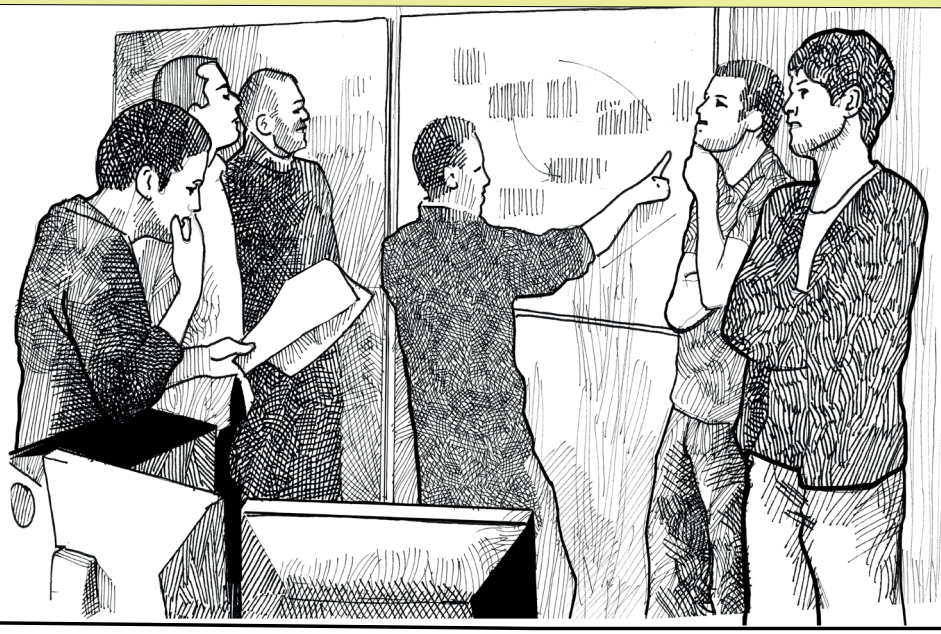
Dato che si segue un'approccio empirico, la pianificazione serve solo per dare un'impostazione di massima a quanto verrà fatto nell'arco di tempo dell'iterazione. Per questo è estremamente importante al termine di ogni iterazione verificare se le ipotesi fatte erano corrette.

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.

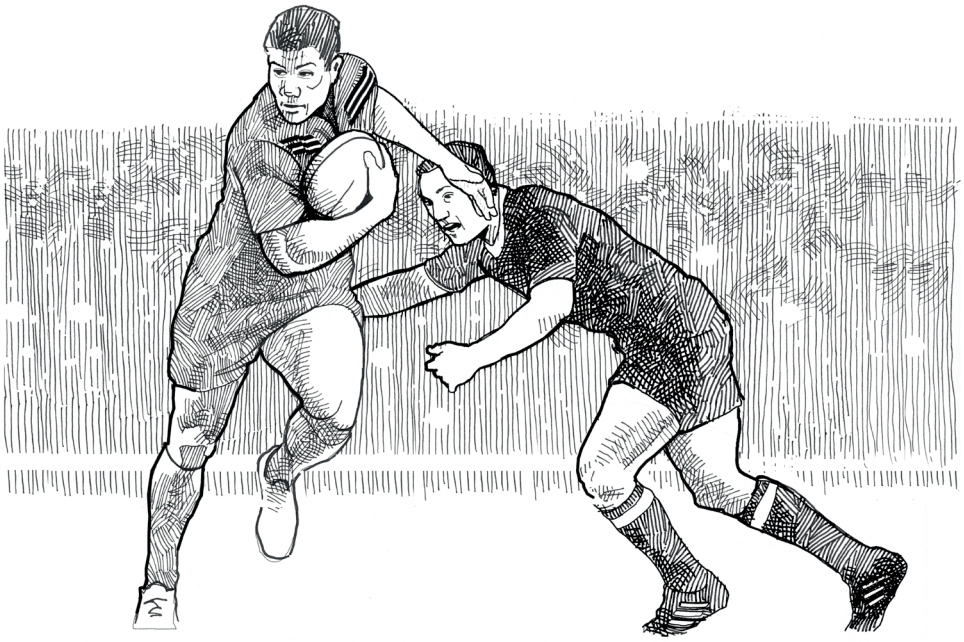
Queste verifiche si svolgono all'interno di altrettanti eventi detti Sprint Review e Sprint Retrospective.



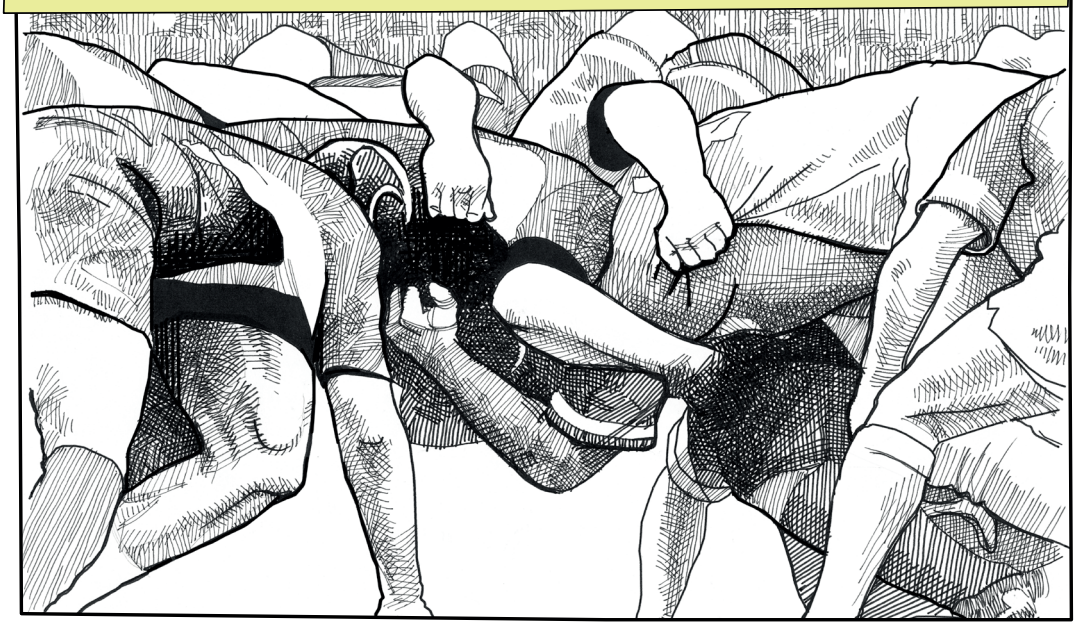
A queste sia aggiunge il Daily Scrum: una riunione quotidiana che si svolge in piedi (per questo è nota anche come Daily Standup Meeting) e che deve durare non più di 10'-15'. Serve per allineare i colleghi sul proprio lavoro e per condividere eventuali difficoltà.

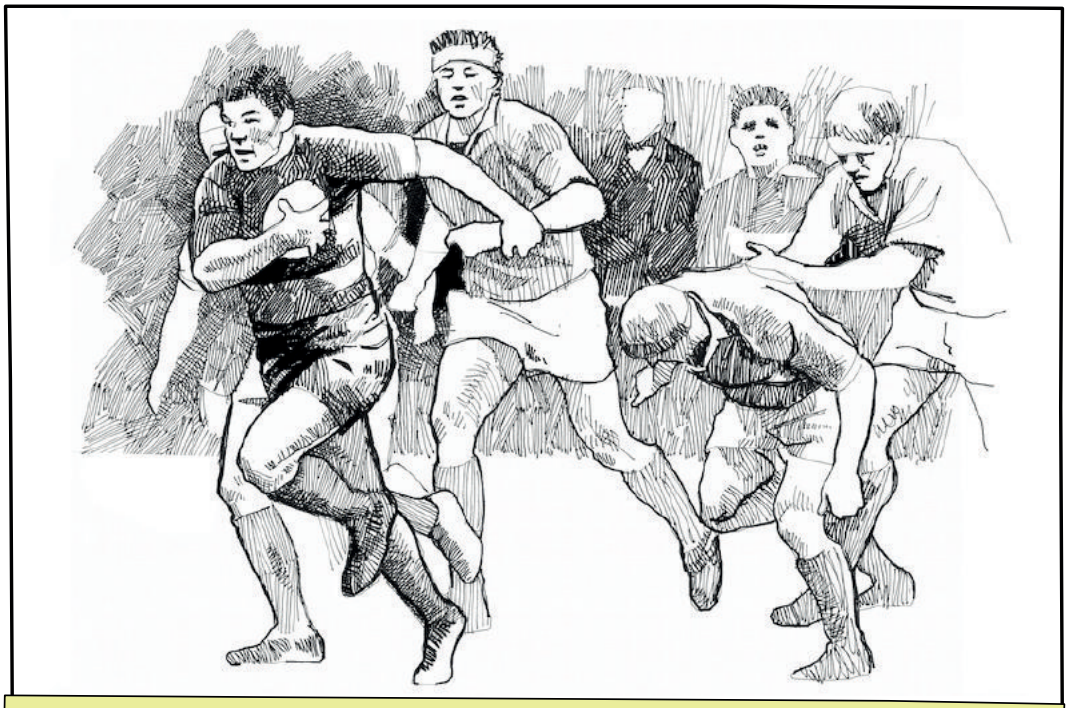


Comandante, lei ha citato spesso il termine Sprint. Cosa significa?



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.

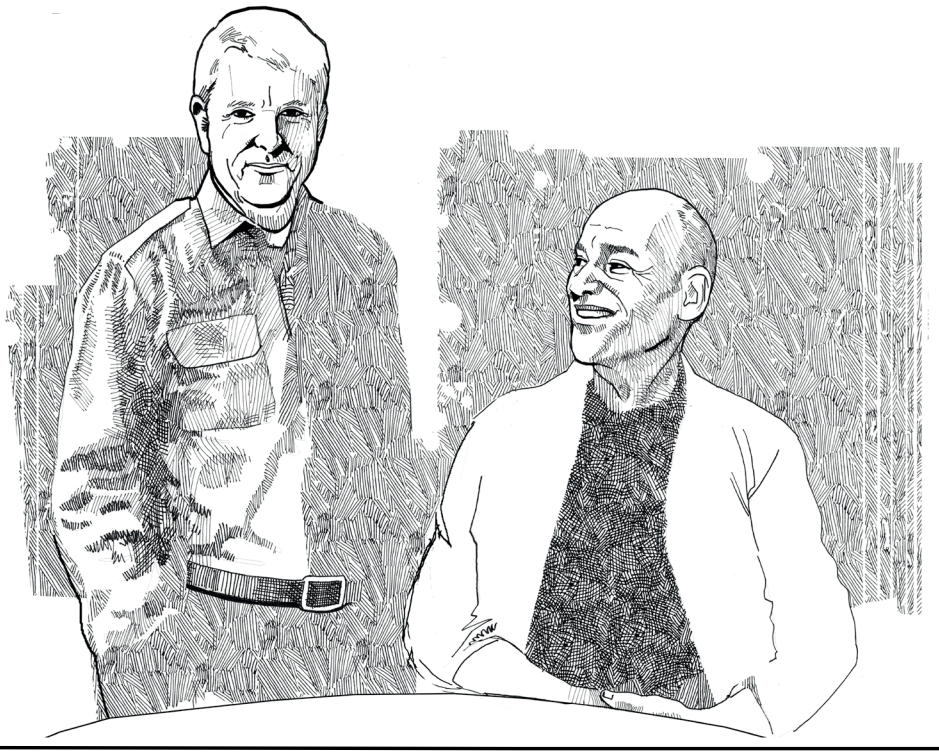




Nel rugby lo sprint è quel tratto di corsa che si fa per portare avanti il pallone. In Scrum, per raggiungere l'obiettivo, ogni membro del team guarda avanti, senza però perdere di vista i colleghi, proprio come nel rugby, dove il giocatore che corre verso l'area di meta deve sempre sapere dove sono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

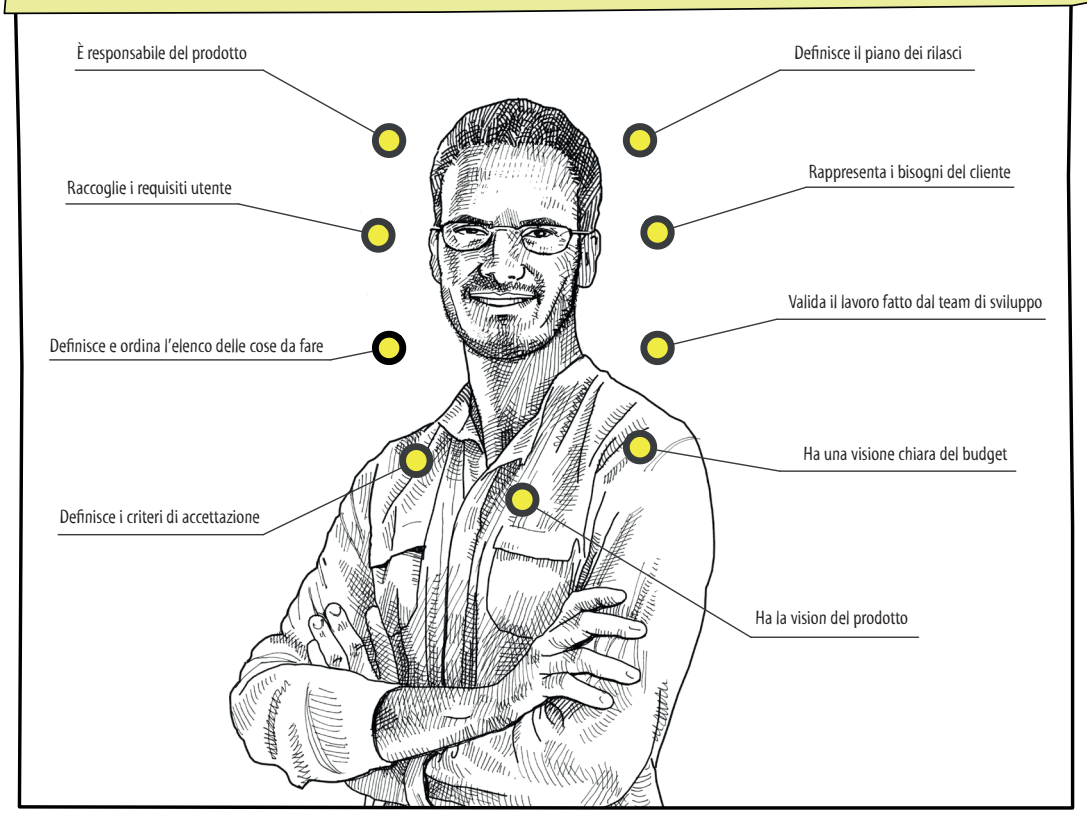
Ken Schwaber e Jeff Sutherland hanno presentato per la prima volta Scrum alla conferenza OOPSLA del 1995. Questa presentazione ha essenzialmente documentato ciò che Ken e Jeff avevano appreso negli anni precedenti dall'esperienza fatta sul campo.



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.

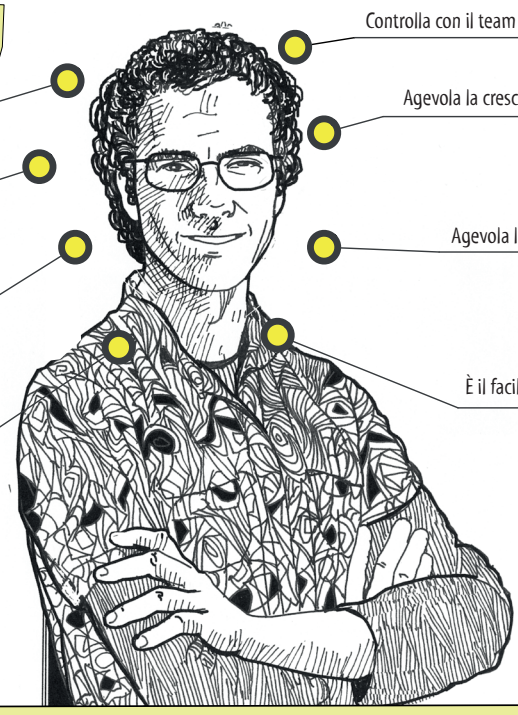
Il Product Owner, abbreviato in PO, è colui che ha la responsabilità del prodotto. Conosce l'utente e i suoi bisogni. È responsabile del cosa sarà contenuto all'interno del prodotto, ma non del come questo verrà realizzato, che è di competenza del team di sviluppo.



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".

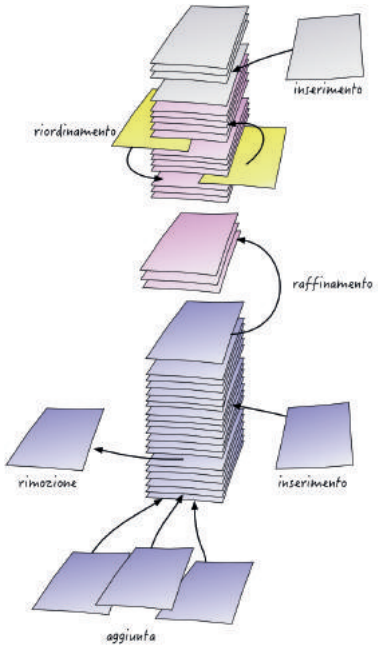
…e lo Scrum Master?



- 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?

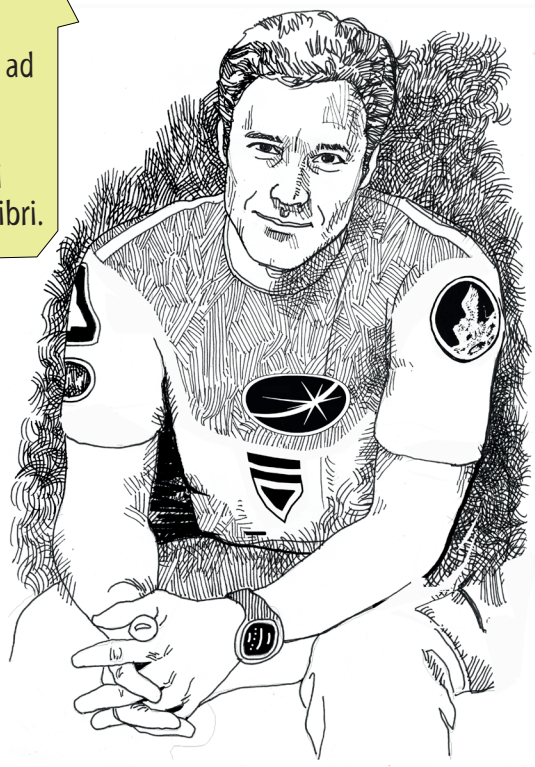


Avremo modo di vederli approfonditamente nel corso della nostra visita. Per adesso sappi che il Product Backlog, il Burn-Down Chart, e per certi versi anche la Task Board, sono importanti strumenti che aiutano il team a tenere sotto controllo il lavoro. Sono molto importanti, ma ricordati cosa è scritto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

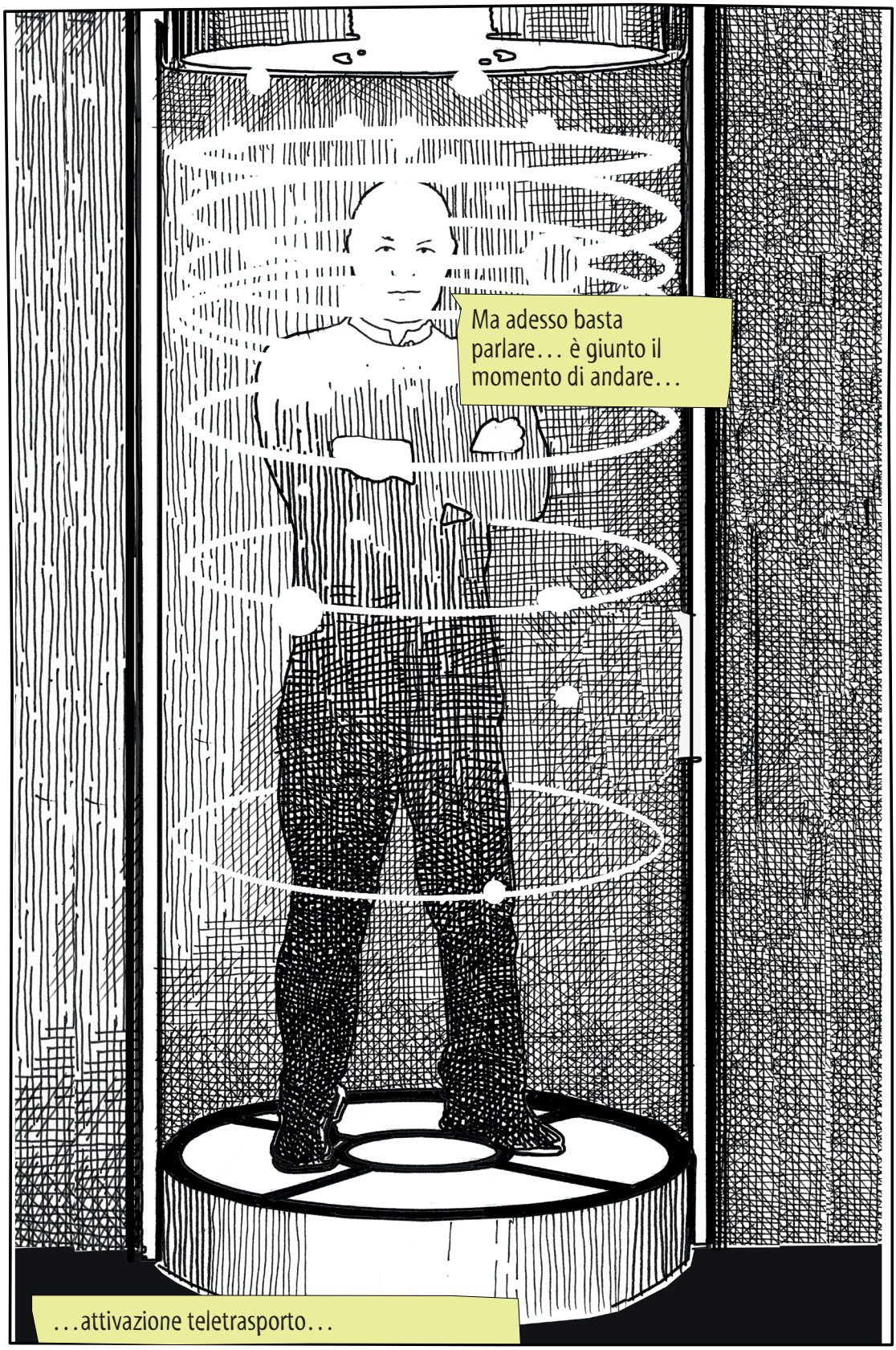
In realtà le regole e le prescrizioni sono poche e puntuali. Scrum può essere vista come una raccolta di best practice formulate sulla base di una lunga esperienza sul campo.

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

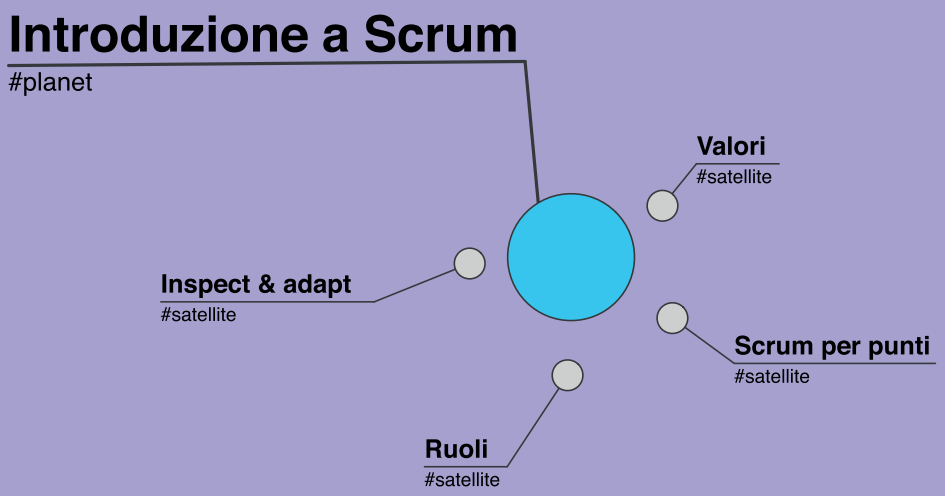


Partendo dai principi e dai valori di Scrum dovrai capire quando quelle raccomandazioni sono utili e quando invece potrai apportare qualche modifica.

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 Ōno (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 Ōno.

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** sarà invece dedicata un'intera parte del libro in una successiva edizione del libro riveduta e ampliata.

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

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.

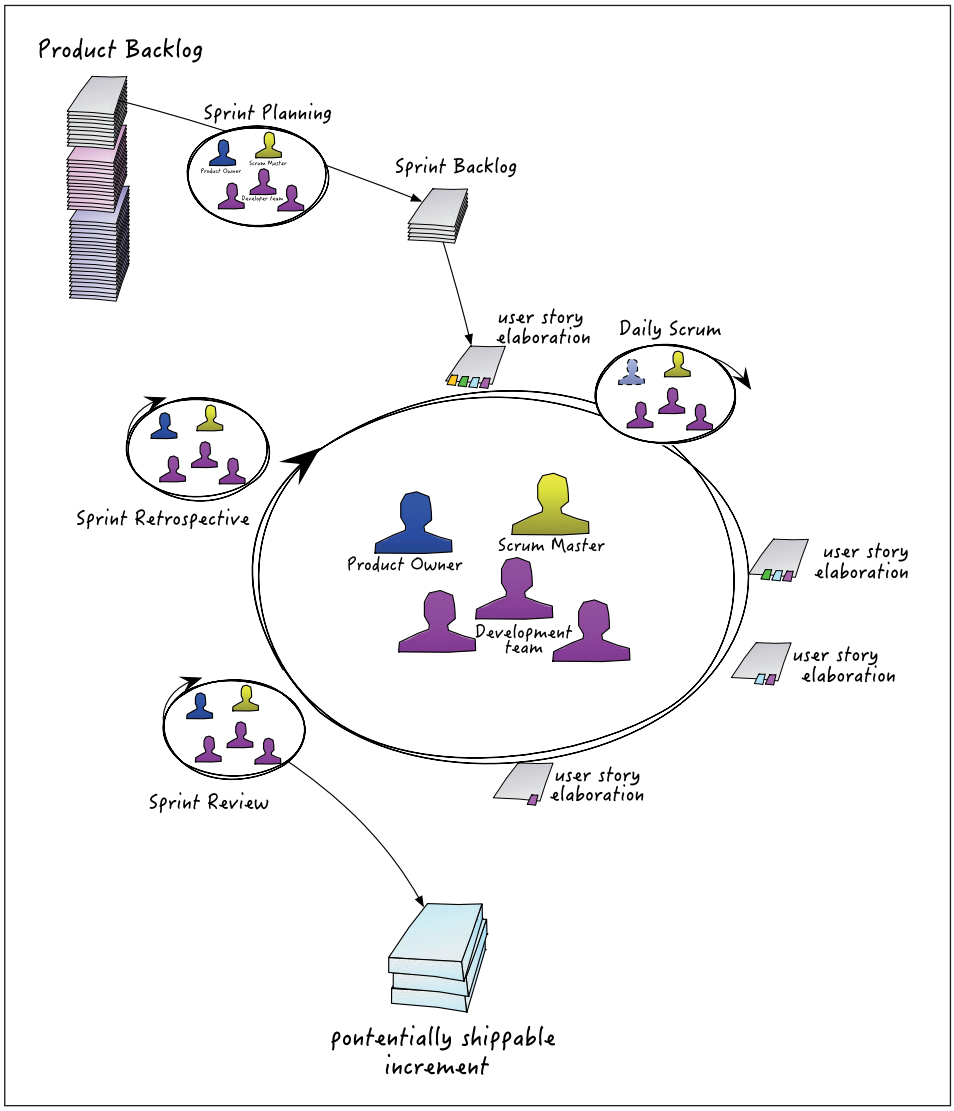


Figura III.1. Diagramma di sintesi di Scrum, framework che si basa su ruoli, eventi e "artefatti".

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. C'è il rischio che due eventi fondamentali come la review e la retrospettiva finiscano per essere svolte proprio in un giorno "prefestivo".

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 **Scrum Master** 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 parlerà in modo dettagliato in una prossima edizione 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. Le retrospettive si basano su ben precisi principi e si possono sviluppare con diversi approcci, in formati anche molto differenziati, per garantire efficacia ed evitare la previdibilità e l'abitudinarietà.

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 (anzittutto 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 è gradualmente scomparsa dal gergo Scrum dove rimane solo *refinement*.

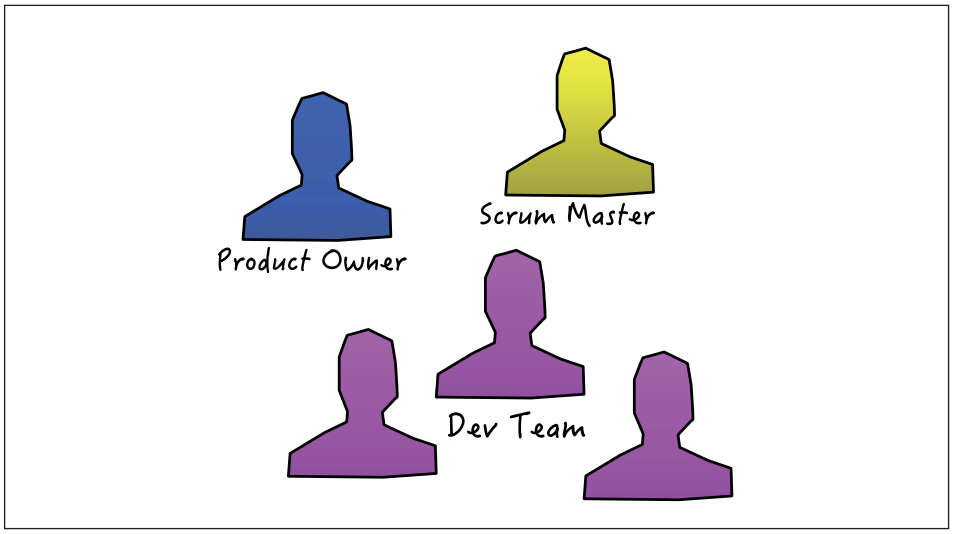


Figura III.2. Un team Scrum è composto da un Product Owner, uno Scrum Master e dal Dev Team.

Ruoli in Scrum

Il framework prevede la definizione di tre ruoli: il **Product Owner**, lo **Scrum Master**, 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 stabilere **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 di quelle esistenti. Il tema della Product Ownership 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.

Scrum Master

Lo **Scrum Master** 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 **Scrum Master** 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.

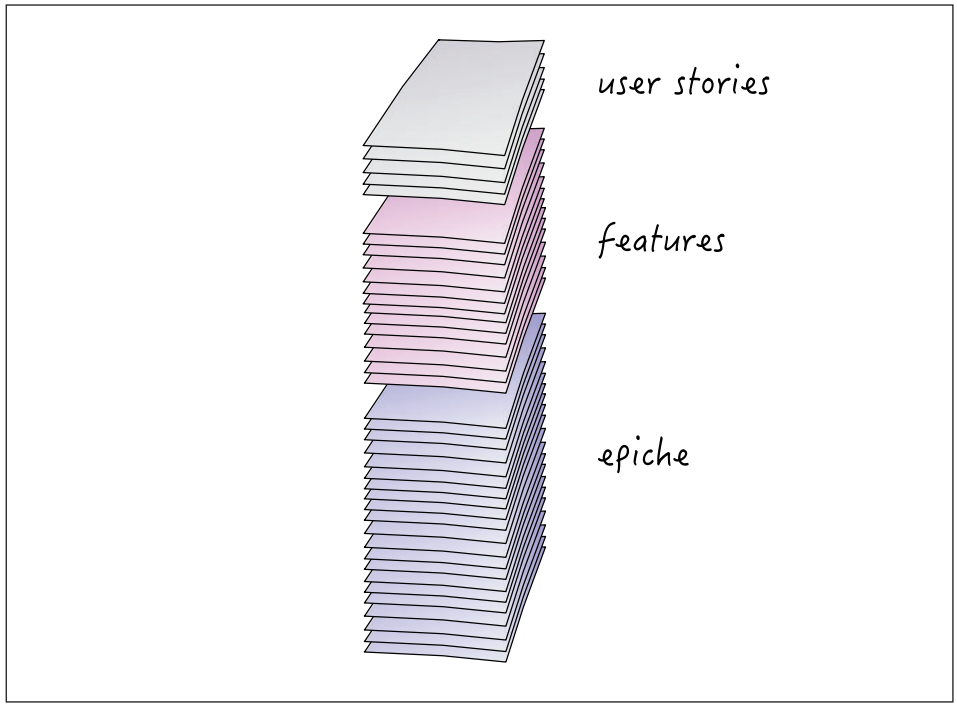


Figura III.3. Il Product Backlog (PB) è una pila di elementi che verranno implementati dal team. In alto, gli elementi già raffinati sono della dimensione giusta per permetterne la lavorazione. In basso, invece, elementi più grossi rappresentano funzionalità o parti del sistema a grana grossa.

"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.

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:

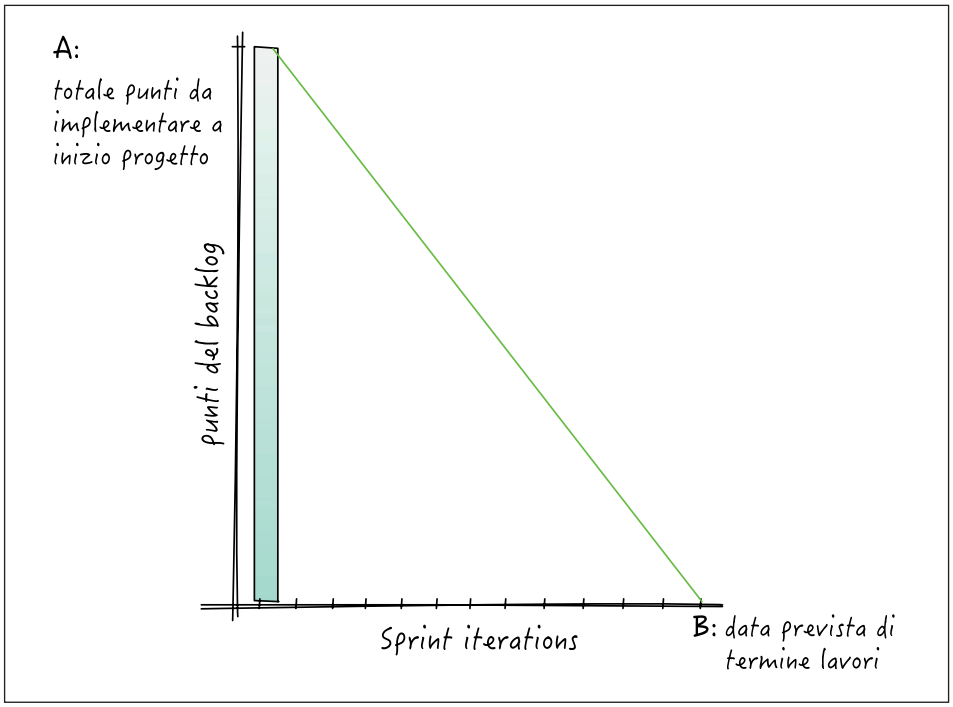


Figura III.4. Il Burn-Down Chart, nella sua fase iniziale.

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**).

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).

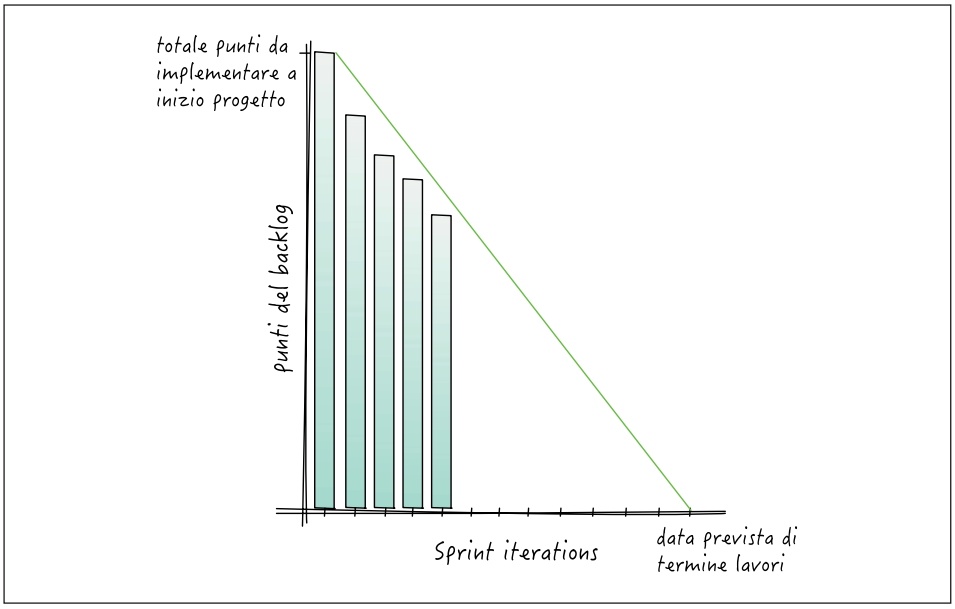


Figura III.5. Dopo qualche iterazione si forma un istogramma che permette di individuare il trend dell'implementazione delle varie storie e consente quindi di capire se la data ipotizzata sarà rispettata.

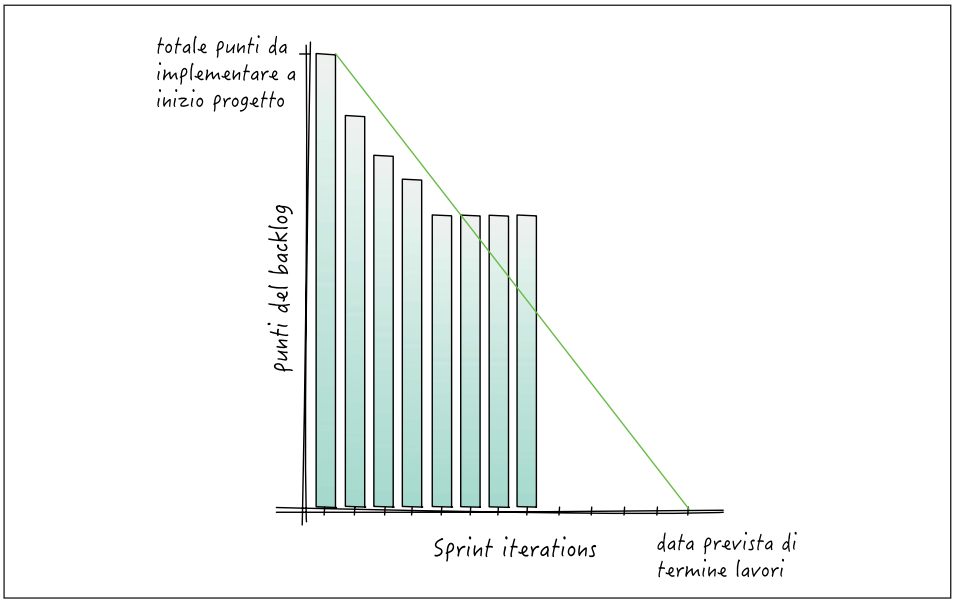


Figura III.6. Il diagramma del Burn-Down visualizza in modo molto efficace se l'attuale trend non permette di rientrare nella data inizialmente stimata.

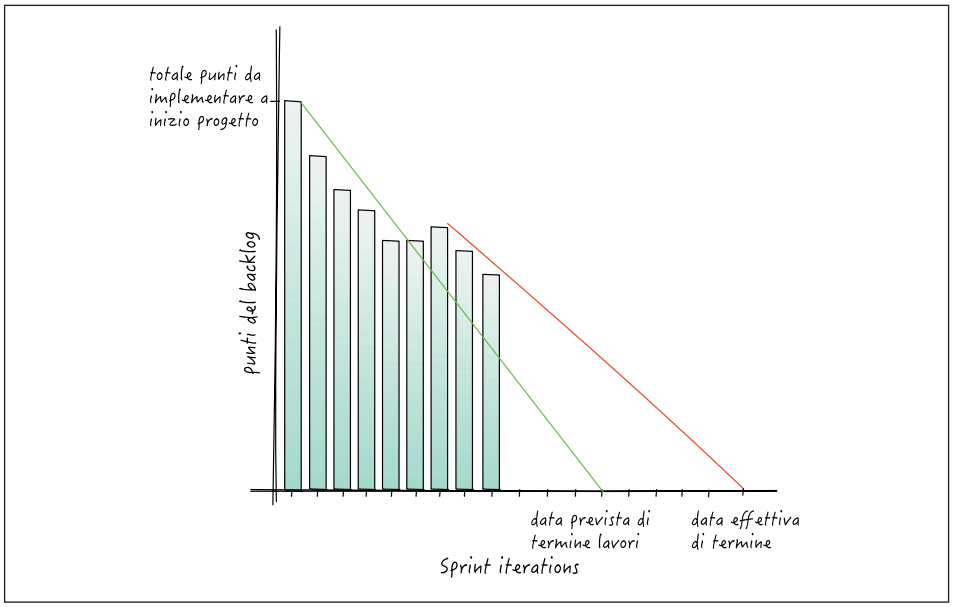


Figura III.7. Se la velocità rallenta, si può ricalcolare la nuova data di consegna, semplicemente tracciando il nuovo trend a zero (in rosso) che interseca i punteggi degli ultimi due o tre sprint.

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 sforamento 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 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.

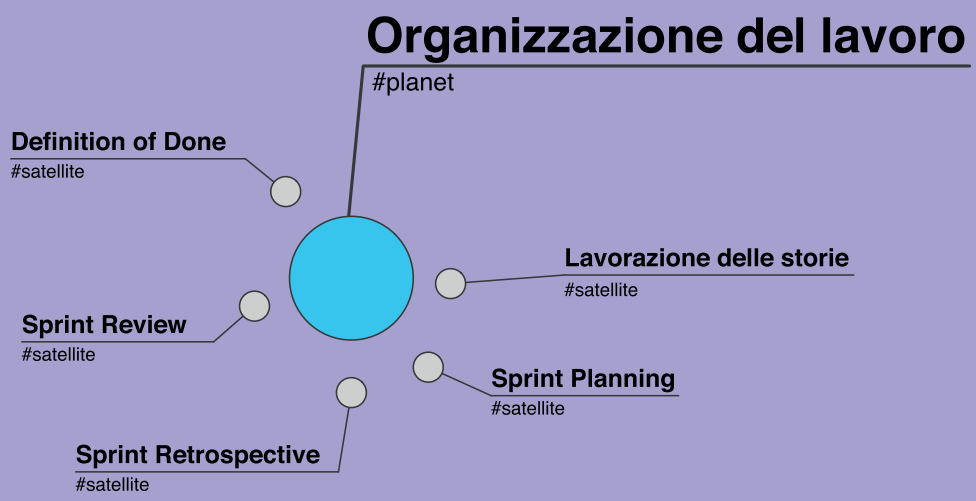
Riferimenti

[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.

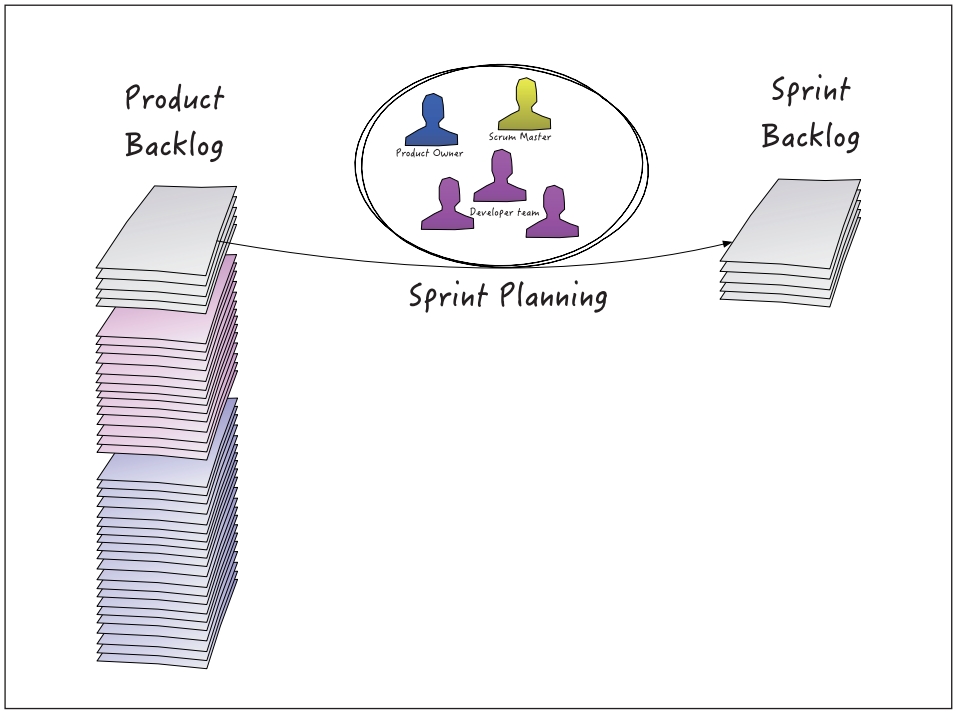


Figura III.8. L'obiettivo dello Sprint Planning è di stabilire quante e quali storie del Product Backlog si potranno implementare nello sprint che sta per iniziare.

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 **Scrum Master** 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**.

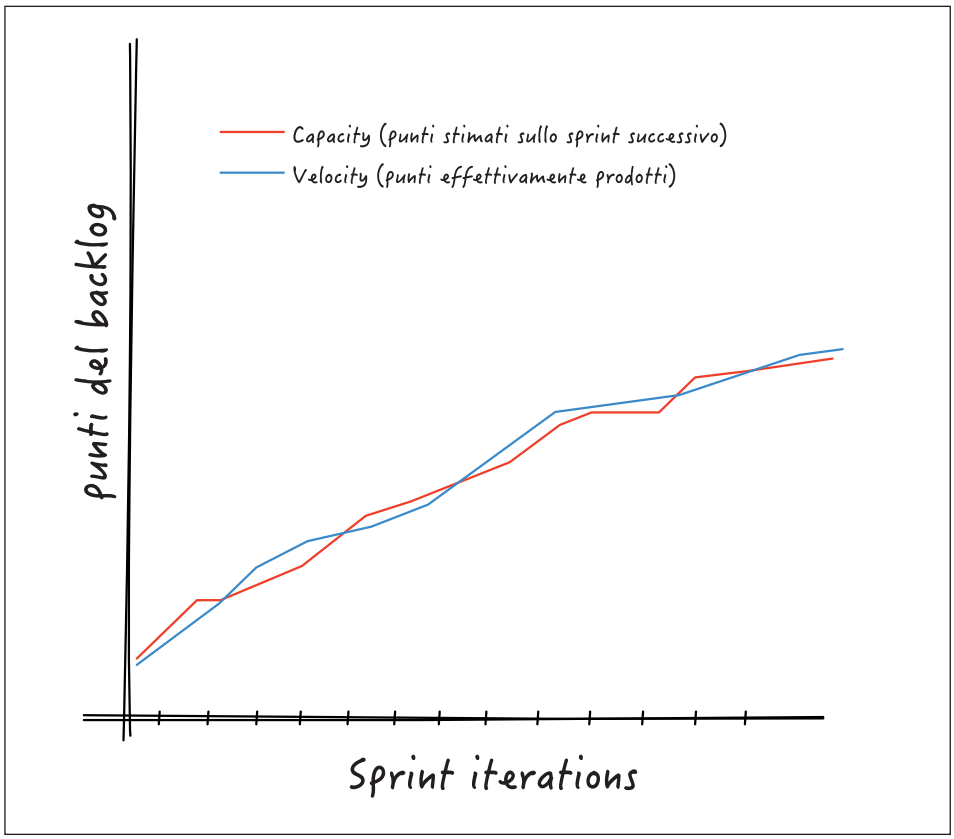


Figura III.9. La velocity del team rappresenta la quantità di cose che il team riesce a implementare in uno sprint. Si misura in punti ed è valutata in modo empirico a posteriori. Il suo valore viene sempre aggiornato, sprint dopo sprint, in modo da avere sempre un valore medio approssimativo affidabile. 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 III.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 Scrum Master 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).

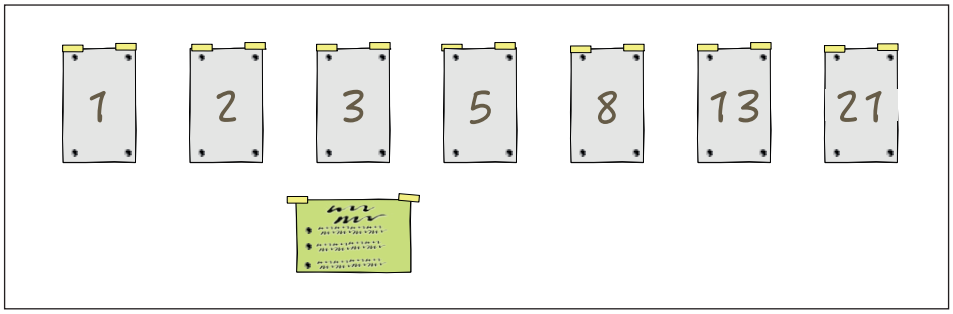


Figura III.11. La prima fase del processo di stima delle storie è identificare una storia di campione, per la quale il team è confidente che il tempo di lavorazione sia dell'ordine di una giornata. Dopo aver disposto le carte con i punteggi su un piano o su una parete, si associa la storia campione alla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

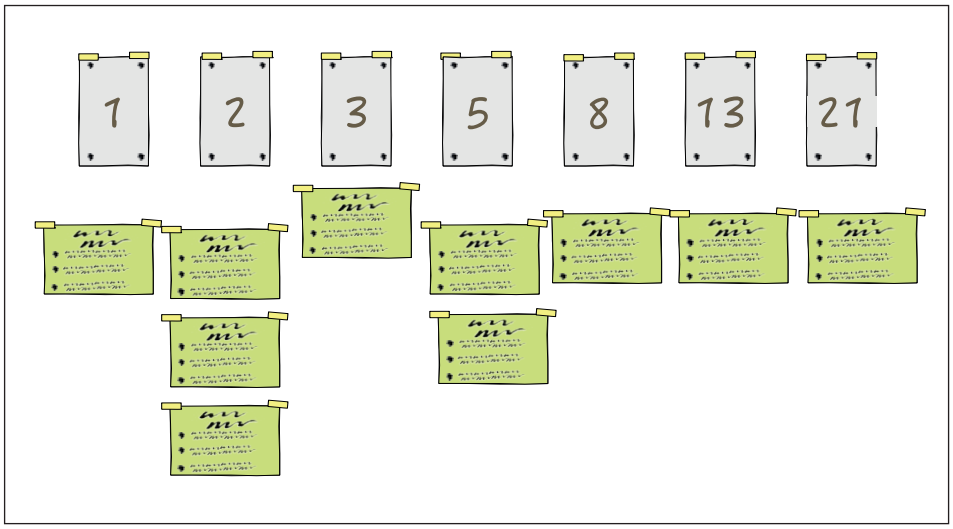


Figura III.12. Successivamente, in base a un processo di confronto, si posizionano le altre storie, cercando di ipotizzare in che rapporto stanno (in termini di effort) rispetto alla storia campione.

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).

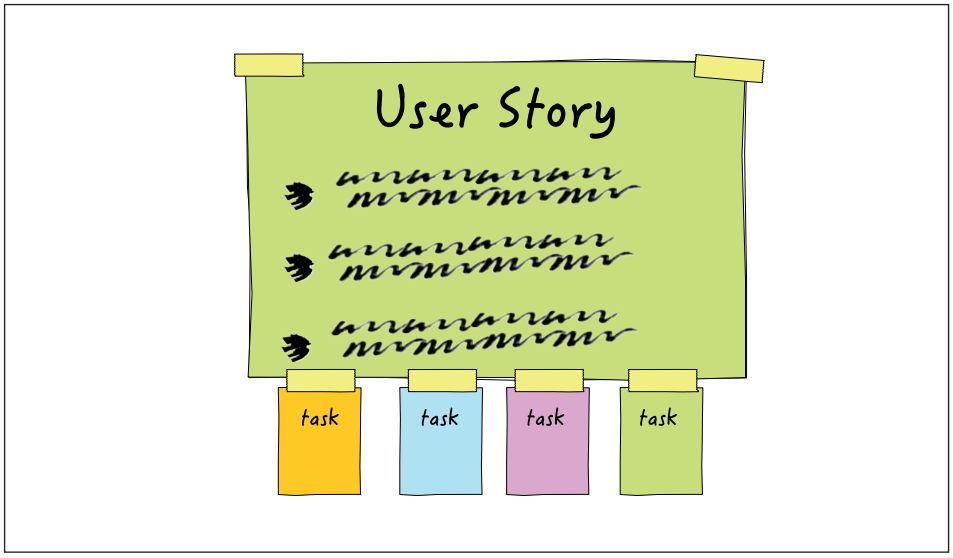


Figura III.13. Scomposizione di una storia in task. Per ogni attività si attacca al cartellino della storia utente un bigliettino più piccolo, con un codice colore concordato.

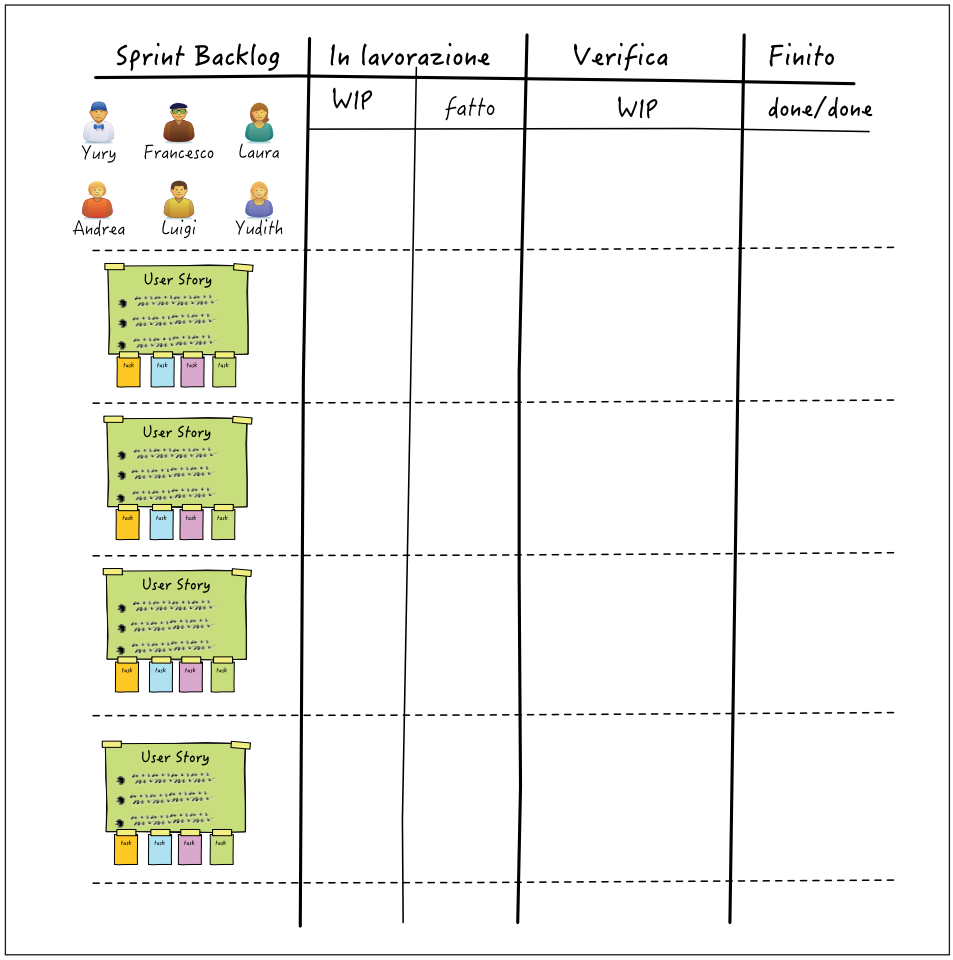


Figura III.14. Le storie sono organizzate su una Kanban board che permette di seguire la lavorazione sia delle sotto-attività che delle storie nel loro complesso.

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.

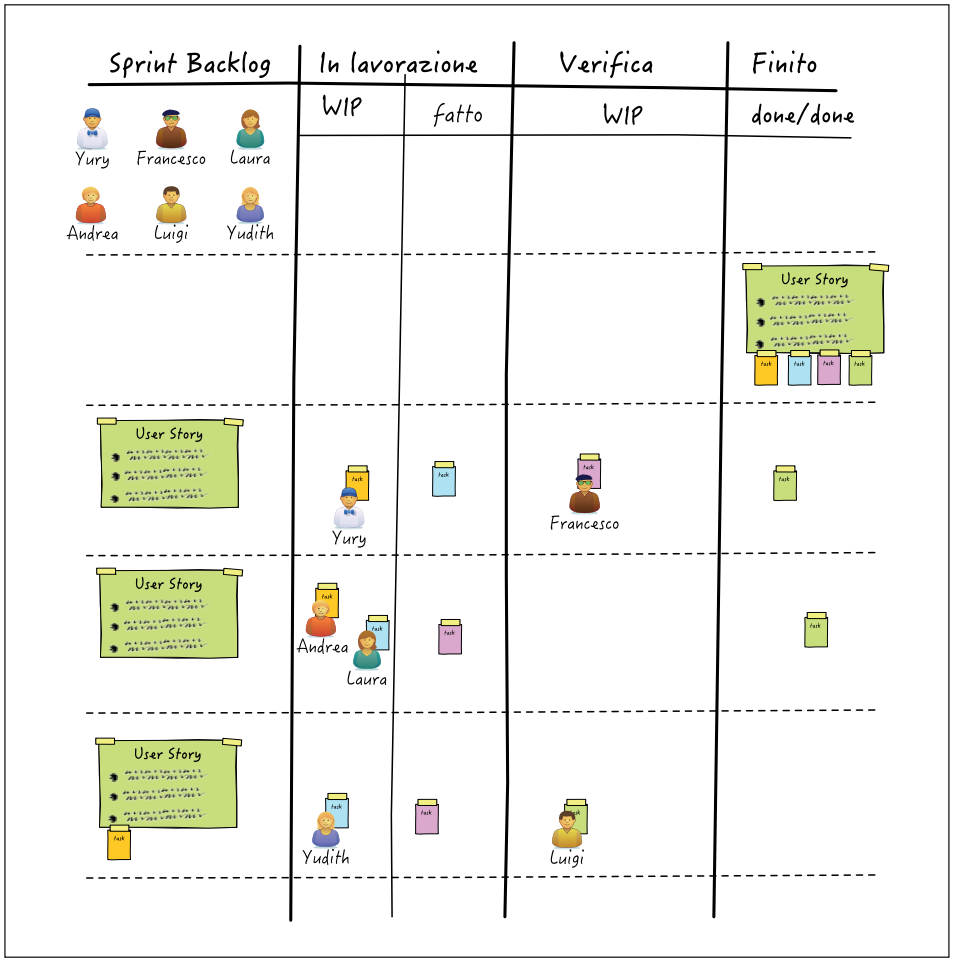


Figura III.15. Grazie all'organizzazione del processo in fasi e sotto-fasi (WIP/done) ogni task può essere parcheggiato, appena si è terminata una fase della lavorazione, e analogamente prelevato per la lavorazione successiva, non appena si libera una persona in grado di eseguire la lavorazione successiva.

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 **Scrum Master** 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 Scrum Master 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 **Scrum Master** 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.

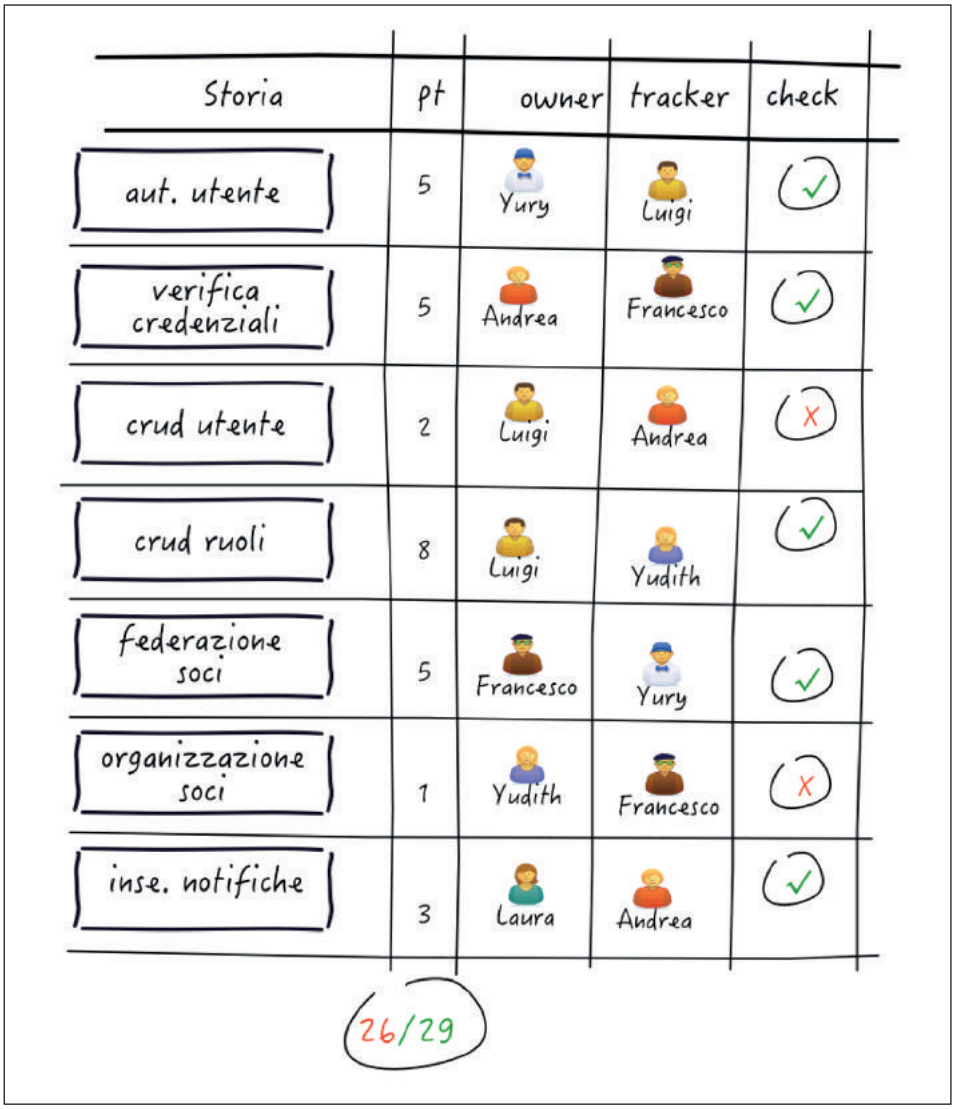


Figura III.16. Per la Sprint Review, lo Scrum Master potrebbe preparare una lavagna con tutte le storie completate in quello sprint e pronte per la demo finale.

- 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 **Scrum Master** 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 nostra intenzione dedicare alle **Retrospettive agili** una specifica parte in una futura edizione riveduta e ampliata di questo libro. In essa saranno presentate le tecniche da utilizzare e le condizioni che le rendono utili per trovare spunti di miglioramento per l'intero team Scrum.

Va inoltre notato che molte delle tematiche e delle tecniche di retrospettiva 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

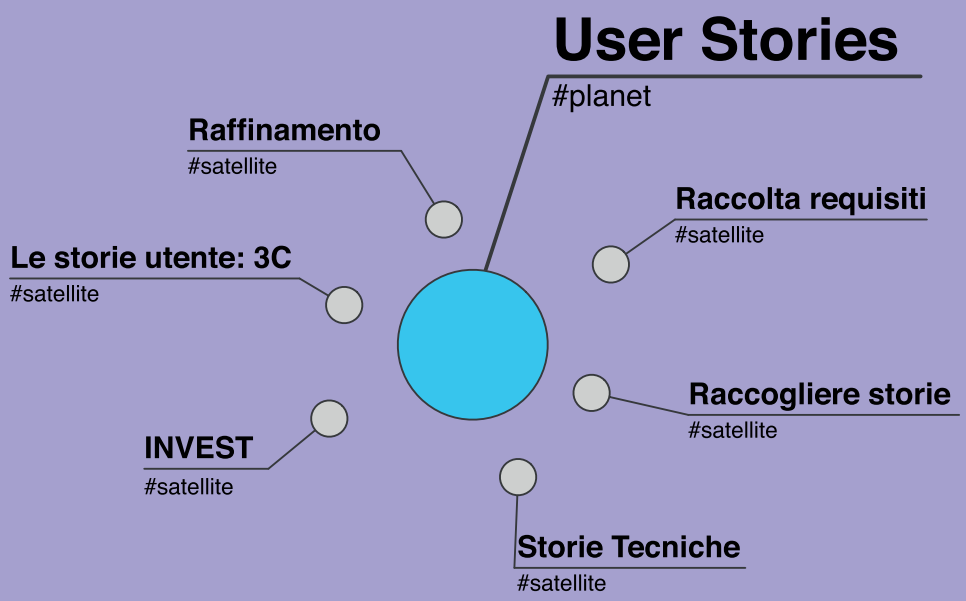
[IS] G. Adzic, D. Evans, *Fifty quick ideas to improve your user stories*. Neuri Consulting LLP, 2014

[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:

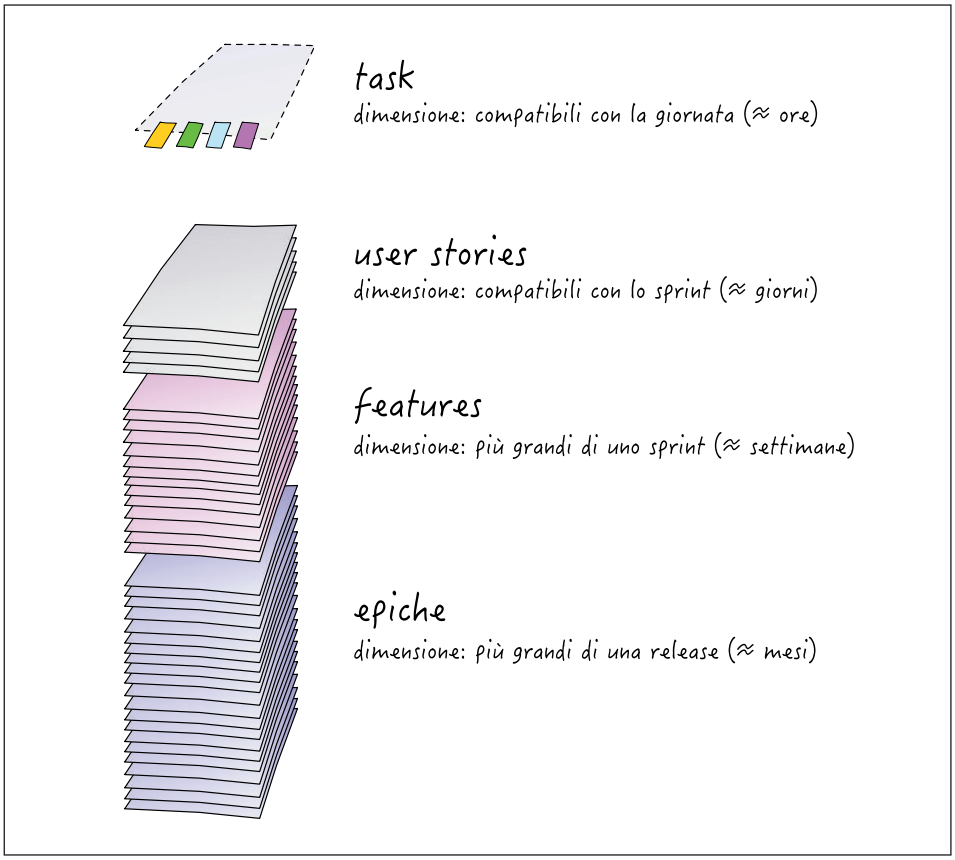


Figura III.17. Il backlog è composto da elementi a grana variabile. Benché siano tutte storie, la grana può variare parecchio.

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

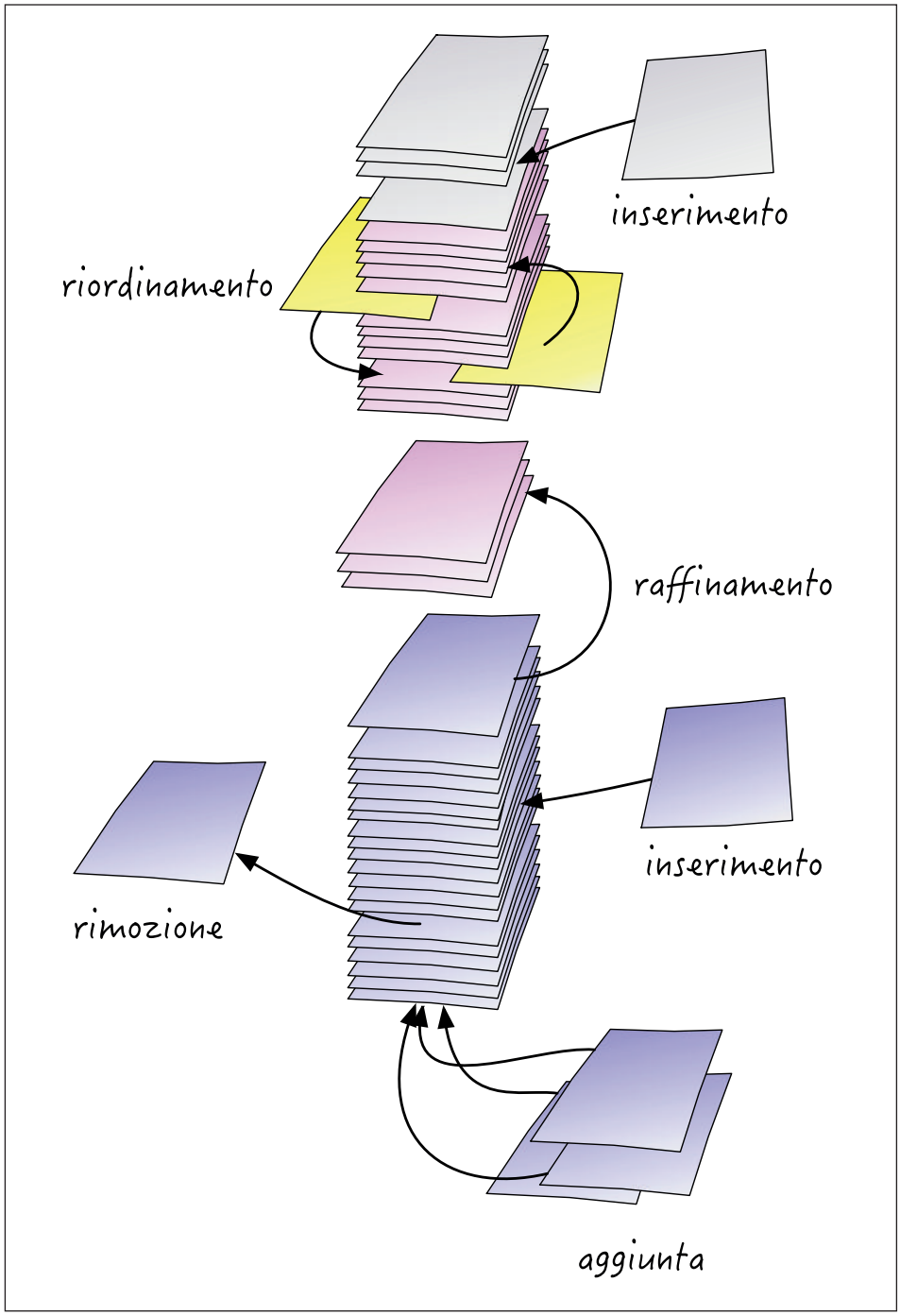


Figura III.18. Le storie che compongono il Product Backlog sono in costante evoluzione, per mezzo del Refinement, sia nella loro struttura che nella dimensione e nel posizionamento.

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).

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

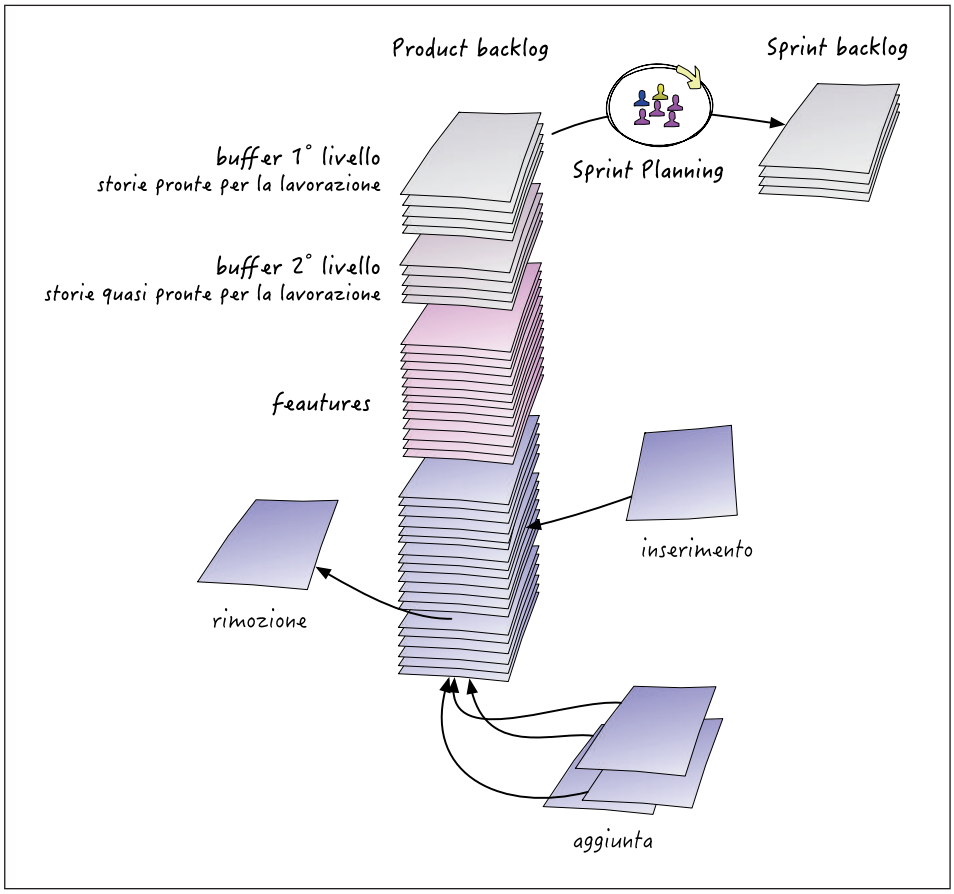


Figura III.19. Il processo di alimentazione del backlog è continuativo. La presenza di uno o due buffer permette di ammortizzare eventuali variazioni in accelerazione sulla velocità di lavorazione del team di sviluppo.

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>

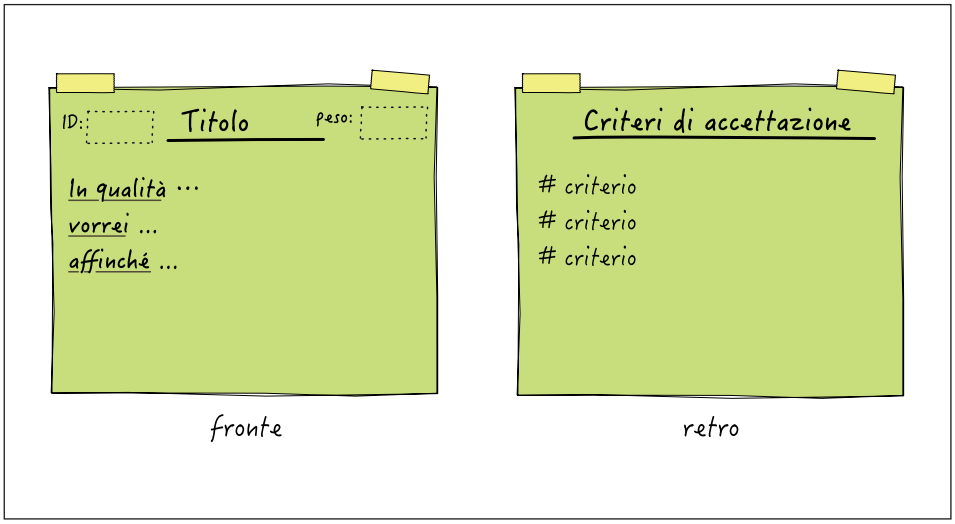


Figura III.20. Un esempio di impaginazione dei due lati di un cartellino secondo uno dei formati più popolari.

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

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 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;
- 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, poiché una **User Story** troppo grande invece potrebbe dar luogo a interpretazioni generiche con ampio margine di variabilità. 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.

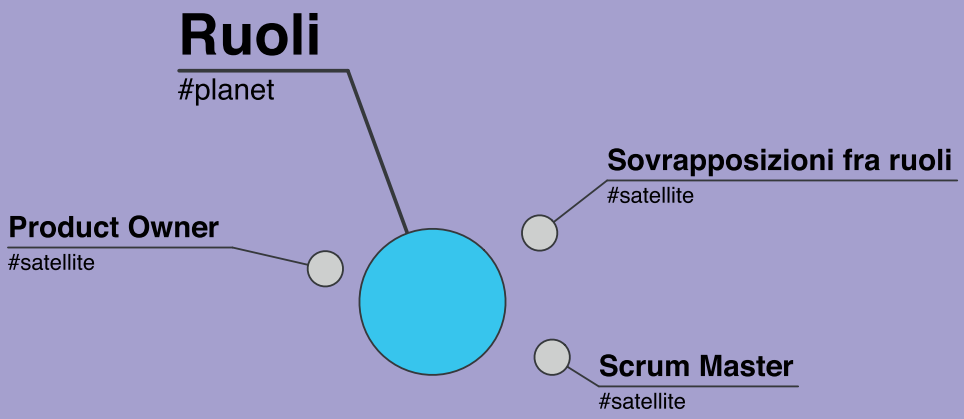
Riferimenti

[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**, **Scrum Master** 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, **Scrum Master** 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 **Scrum Master** 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 Scrum Master

Lo **Scrum Master** (**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 Scrum Master 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 Scrum Master. 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 Scrum Master 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 Scrum Master 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 **Scrum Master** 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 **Scrum Master**, 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 **Scrum Master** 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 Scrum Master. Parte seconda: essere il coach del gruppo

Secondo la visione del ruolo dello **Scrum Master** vista nel paragrafo precedente, lo **Scrum Master** 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 **Scrum Master** 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 **Scrum Master** 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 **Scrum Master** 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 Scrum Master

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 **Scrum Master**. 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 **Scrum Master** 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.

Scrum Master: 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 **Scrum Master** 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, Scrum Master), 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

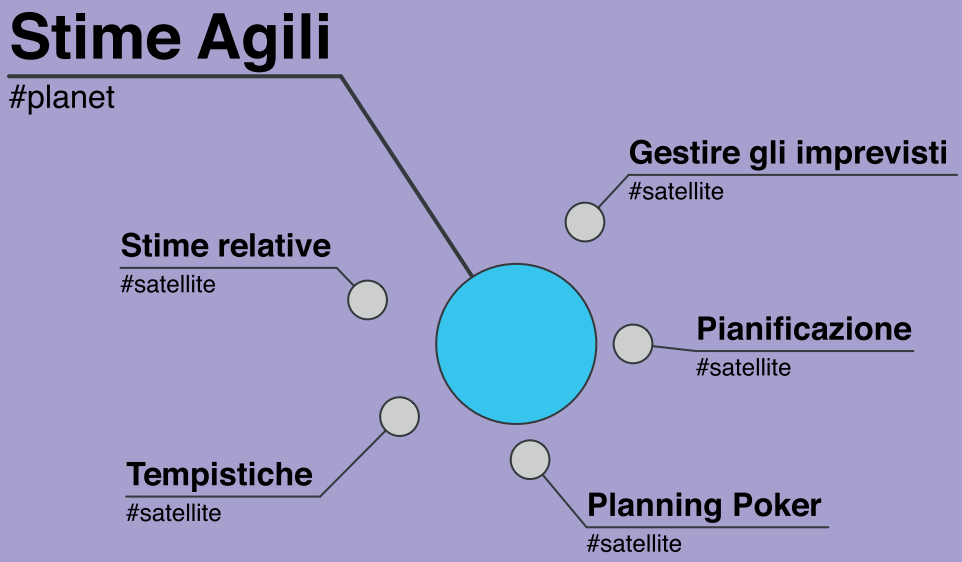
[AJE] La pagina Wikipedia sul libro di Hermann Hesse *Il pellegrinaggio in Oriente* (1932)
https://it.wikipedia.org/wiki/Il_pellegrinaggio_in_Oriente

[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

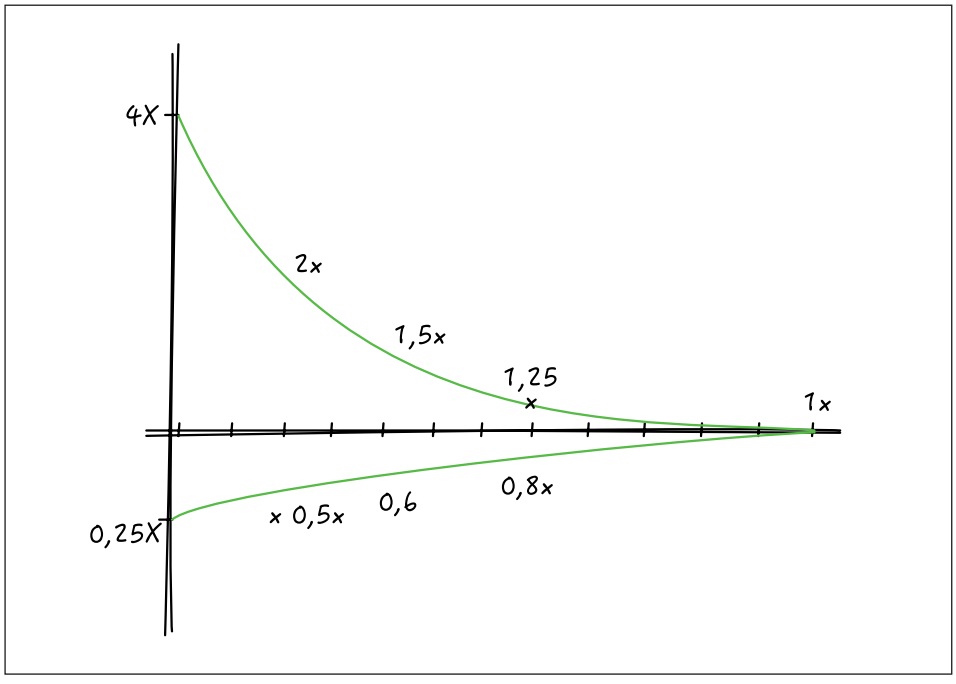


Figura III.22. Diagramma detto del "cono dell'incertezza": partendo da uno stato iniziale in cui la variabilità delle stime passa da -1/4 al quadruplo, man mano che il tempo passa e le informazioni arrivano, si può ridurre questa forbice.

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

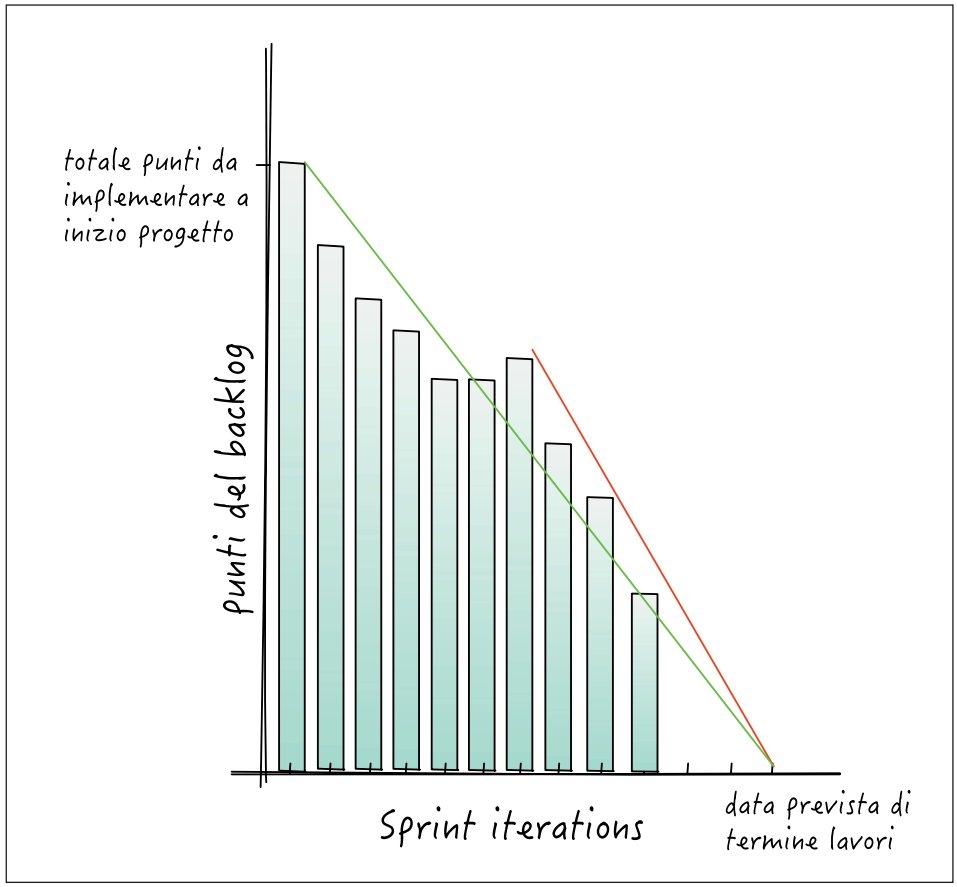


Figura III.23. Se l'attuale trend non permette di rientrare nella data inizialmente stimata, una soluzione potrebbe essere quella di aumentare la velocity di sprint.

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.

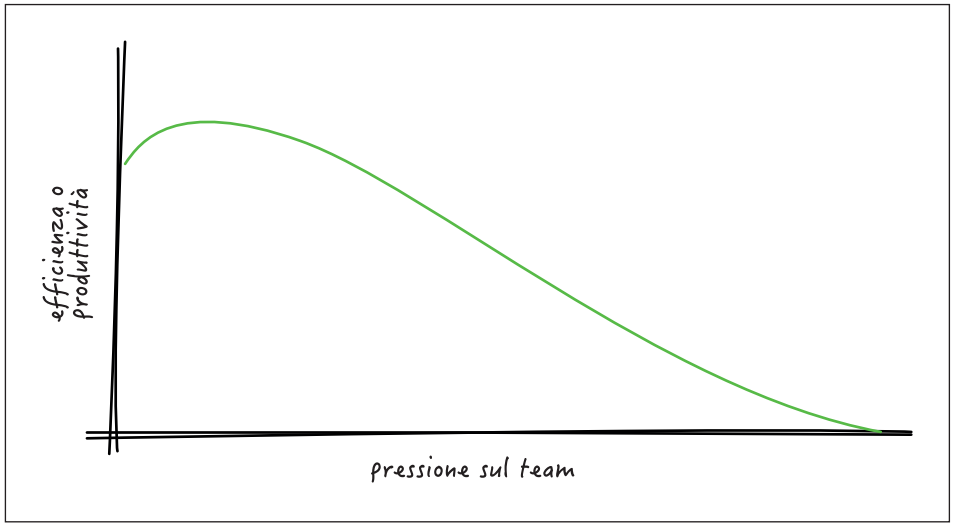


Figura III.24. Andamento della qualità del lavoro svolto in relazione alla pressione del sistema e dello stress sulle persone: a volte aumentando di un poco la pressione sul team le persone lavorano in modo più concentrato e quindi producono meglio. Continuando ad aumentare la pressione, però, la produttività 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.

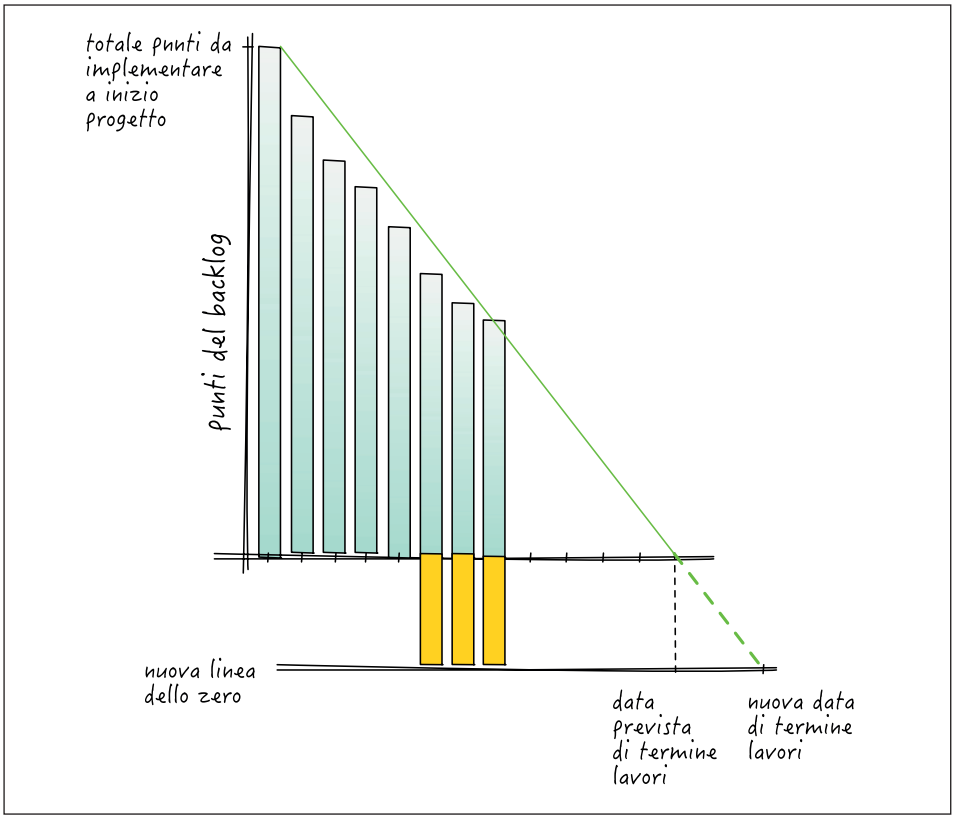


Figura III.25. Il cliente, nel mezzo del progetto, chiede di aggiungere nuove funzionalità al prodotto finale. In questo caso spesso le nuove funzioni si aggiungono sotto la linea delle ascisse e si punta a considerare terminati i lavori quando si arriverà alla nuova linea dello zero.

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.

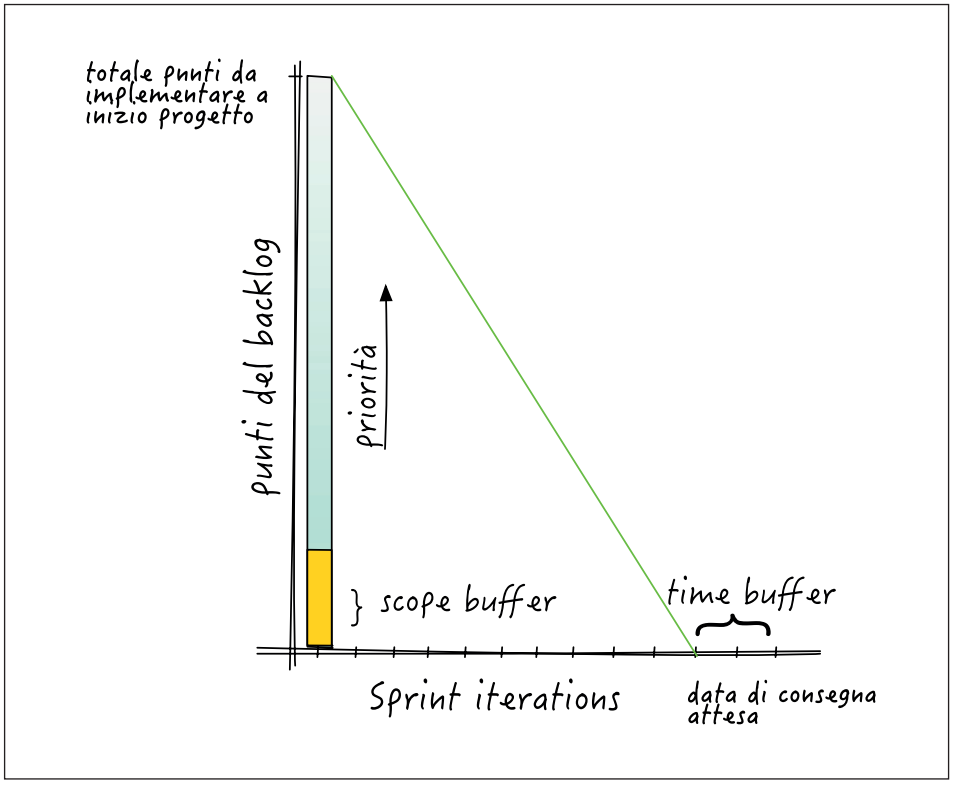


Figura III.26. La gestione della release di un prodotto agile passa per la gestione di aree di variabilità sia sulle tempistiche (time buffer) che sulle funzionalità di implementare (scope buffer).

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/

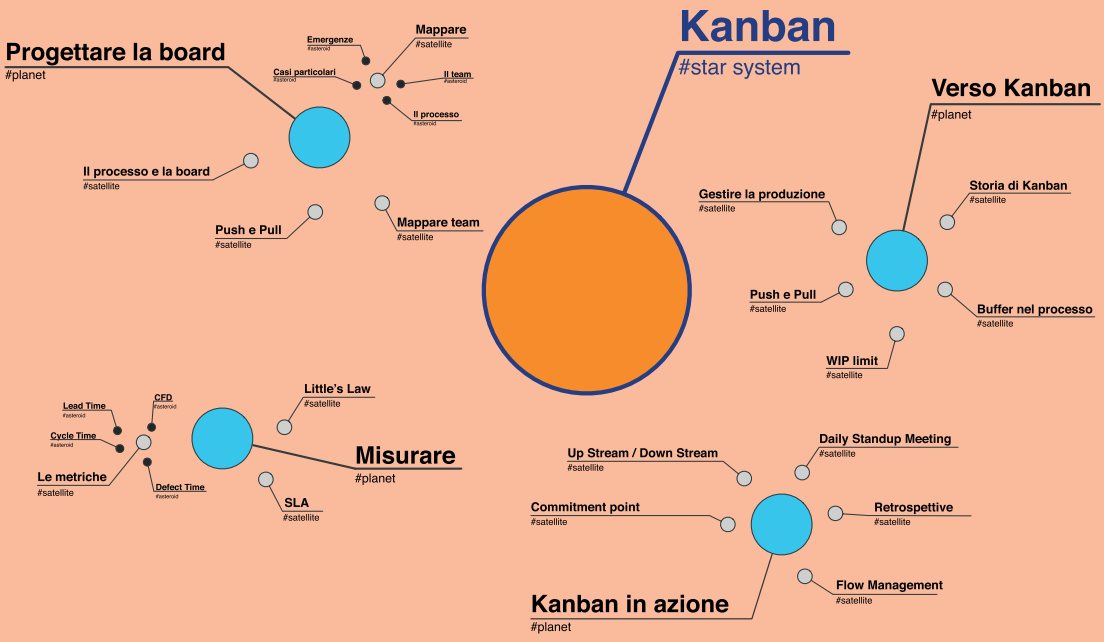
[MF] Martin Fowler, "PurposeOfEstimation"
http://martinfowler.com/bliki/PurposeOfEstimation.html

[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

PARTE IV
KANBAN

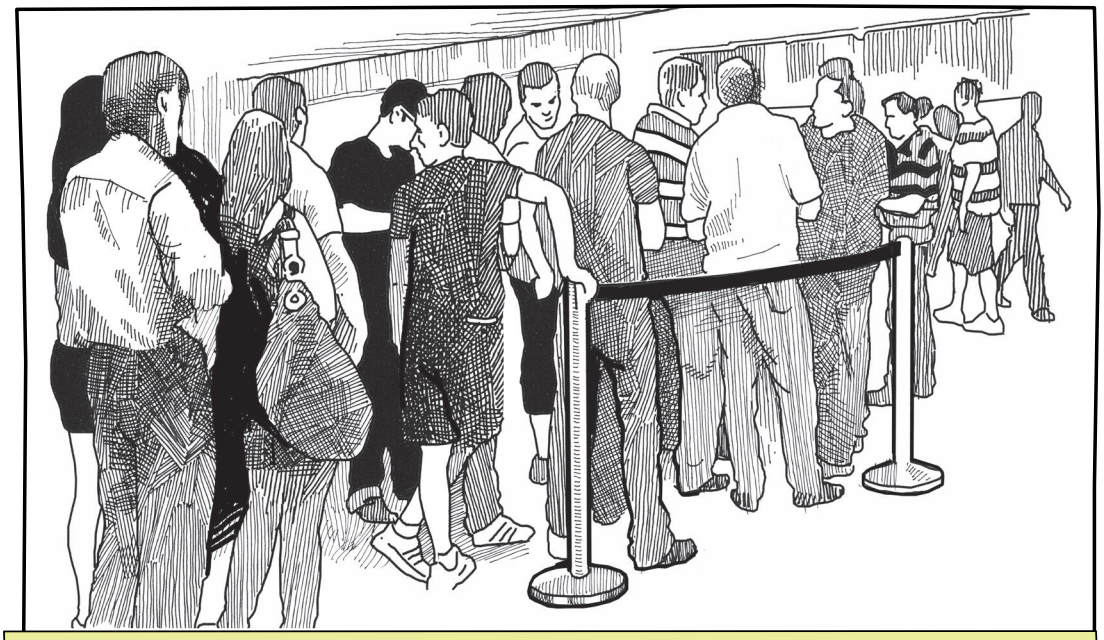
Giovanni Puliti



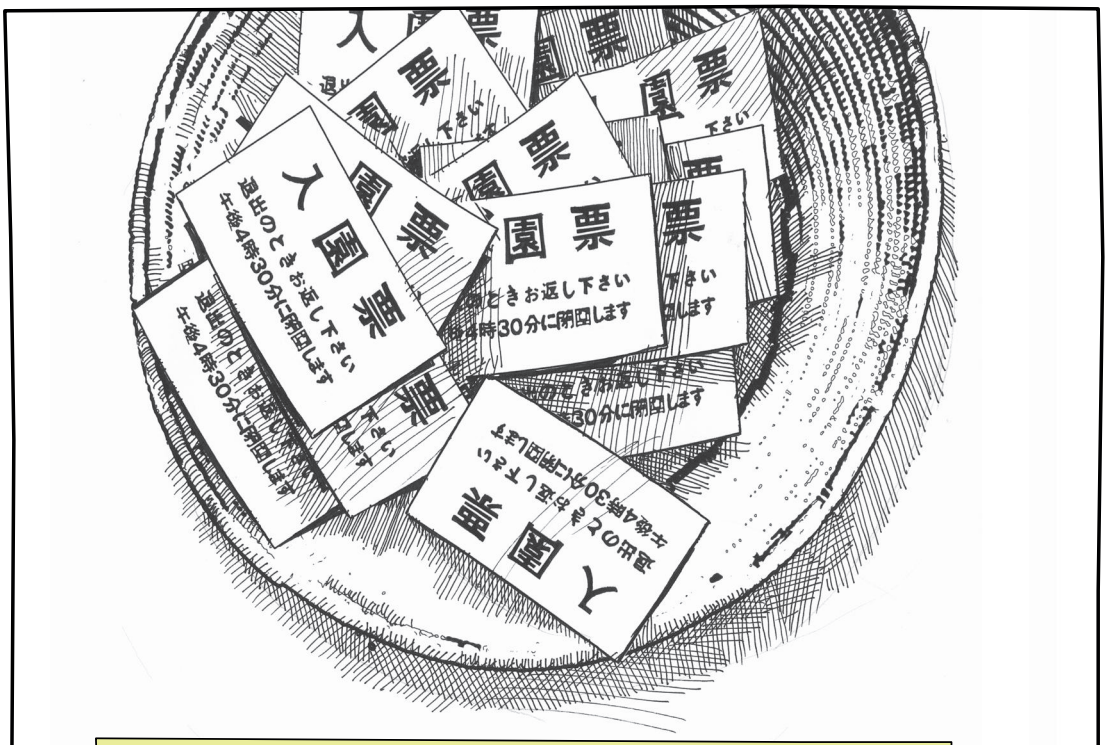
Parte 4
Il sistema Kanban

Nella lingua nativa del Giappone, "kanban" significa "segnale visivo" e la leggenda narra che la sua invenzione si deve ai guardiani del giardino imperiale giapponese di Tokyo… sul pianeta Terra.

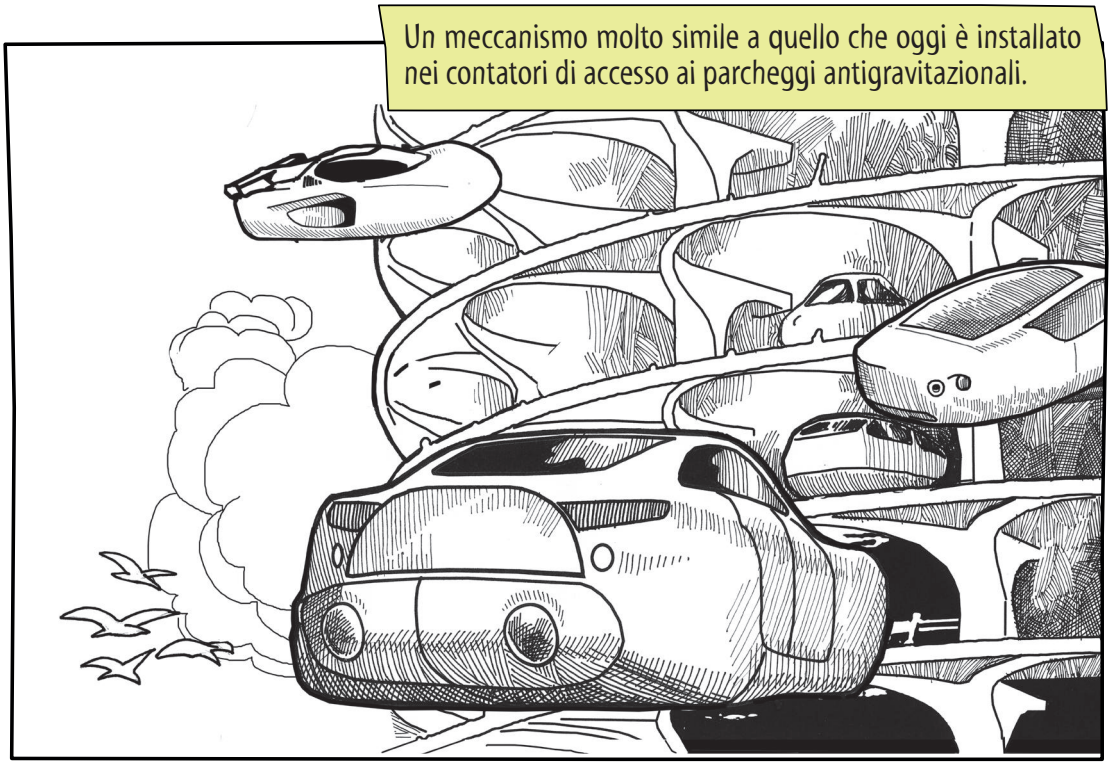
Per tenere sotto controllo il numero dei visitatori, fu inventato un meccanismo molto semplice per gestirne il flusso.



All’entrata, ogni visitatore doveva prelevare da un contenitore una piccola tessera intarsiata. All’uscita, doveva restituire tale tessera. Quando non vi erano tessere nel contenitore, nessun altro visitatore poteva entrare.

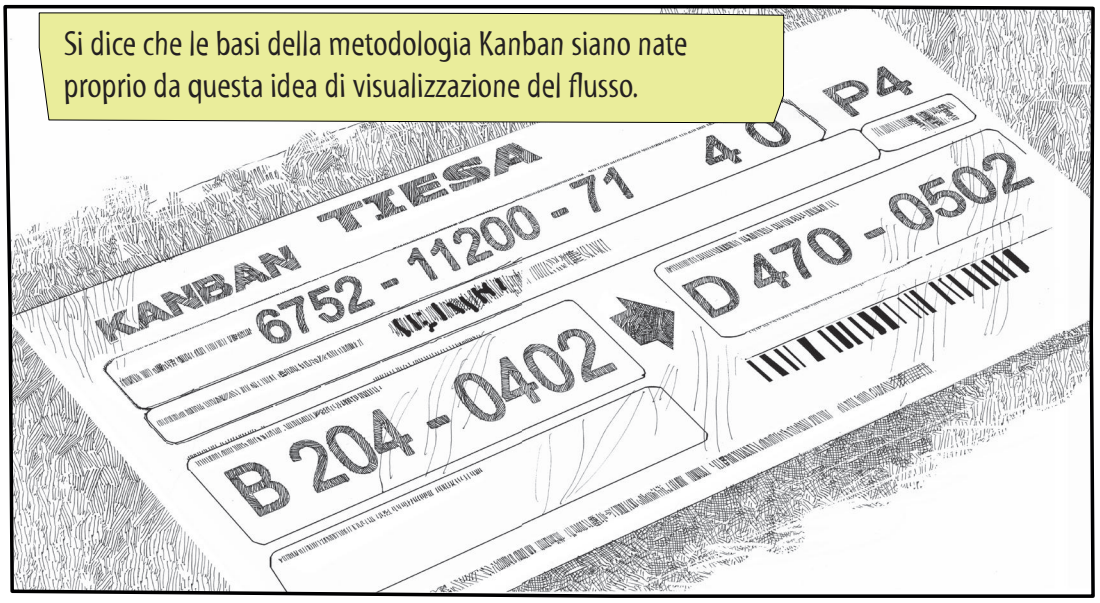


Il numero di tessere nel cestino corrispondeva al numero massimo di visitatori ammessi al giardino.


Un meccanismo molto simile a quello che oggi è installato nei contatori di accesso ai parcheggi antigravitazionali.


E cosa c'entra tutto questo con la metodologia Kanban?

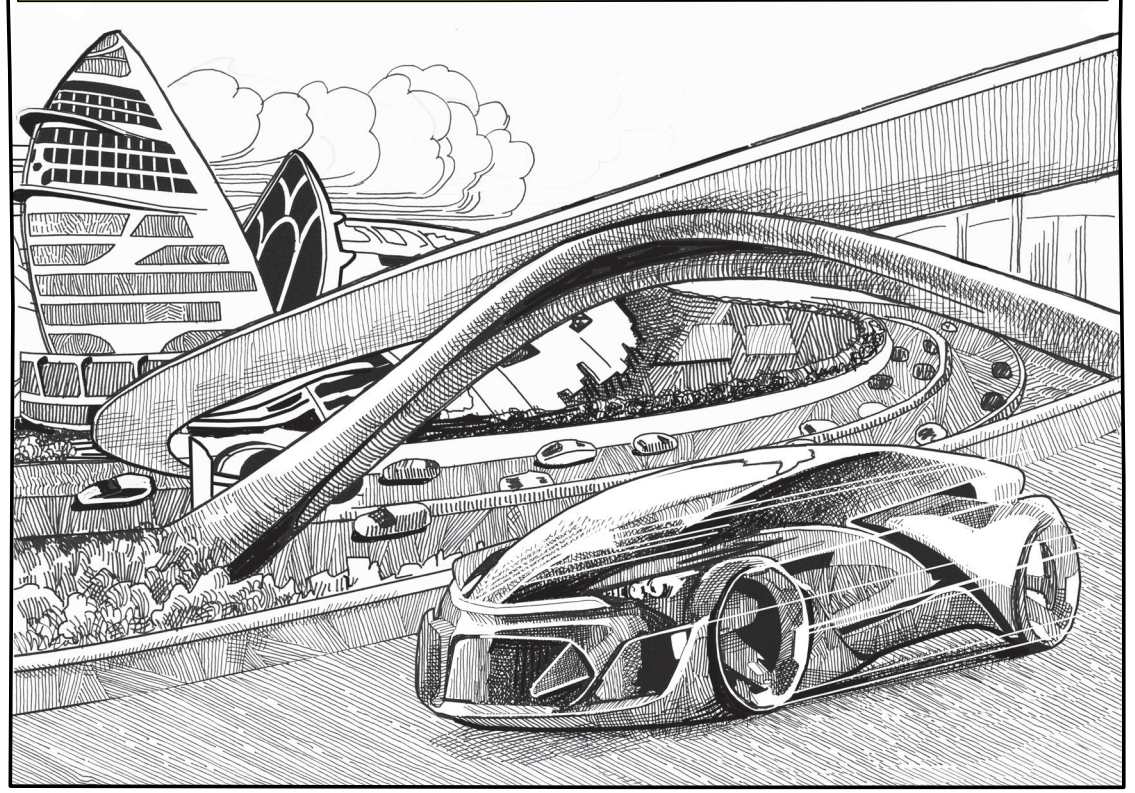
Si dice che le basi della metodologia Kanban siano nate proprio da questa idea di visualizzazione del flusso.

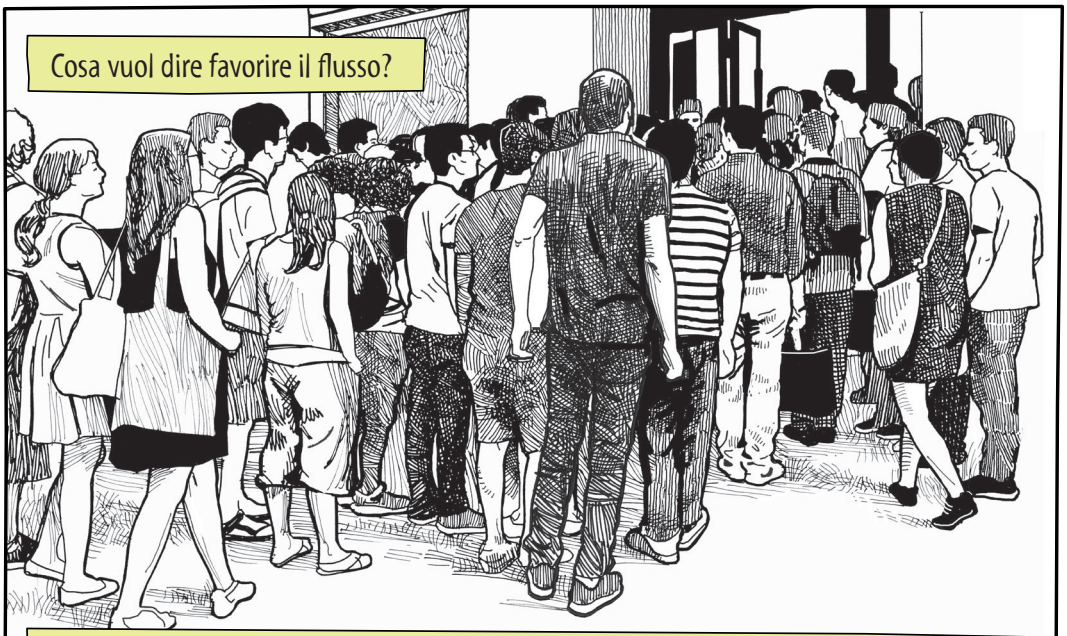


Per esempio, rifacendosi al concetto di transito dei visitatori, si è presa coscienza che le prestazioni di un sistema di produzione sono ottimali se si evita di raggiungere la saturazione e al contempo si velocizza il flusso di scorrimento.

È un fenomeno che i terrestri avevano sperimentato sulla loro pelle passando molte ore bloccati in coda nelle loro auto a combustibili fossili…

I principi messi in atto nel giardino imperiale giapponese sono alla base dei regolamenti delle moderne linee di collegamento che percorriamo tutti i giorni con i nostri sprinter: aumentare la banda (numero di mezzi che possono transitare in un istante) non basta; per esempio è necessario ridurre la velocità media e possibilmente semplificare la viabilità per favorire lo scorrimento.





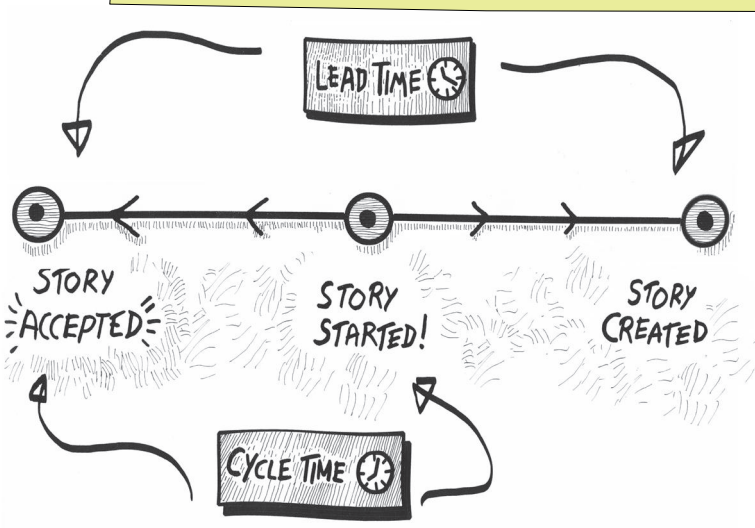
Significa, per esempio, cercare di impedire l'insorgere di ingolfamenti nel percorso, qualunque esso sia, per non incorrere in un rallentamento o peggio ancora in un blocco. Vale per una coda di persone, per una autovia o per un processo di produzione all'interno di una moderna fabbrica.

Grazie all'uso delle tecniche Kanban, gli esperti di processi di produzione si resero conto che era utile semplificare il processo in modo da stabilizzarlo. Un noto adagio diceva: meglio una lavorazione lenta ma costante, che una che procede per sbalzi di velocità.

È quello che succede in una scala mobile che procede lentamente, ma senza soluzione di continuità e senza sbalzi.

Con questo approccio si punta a fare meno cose in parallelo, ma a velocizzarne la lavorazione.

Una celebre frase dice: "Stop starting, start finishing".



Minore è il tempo di lavorazione, maggiore sarà la prestazione complessiva.

Prima di intervenire è necessario osservare il flusso all'interno del processo di produzione del sistema

Misurare e gestire il flusso

Esplicitare le regole del processo Identificare le possibili opportunità di miglioramento.

Limitare il Work-in-Progress (WIP)

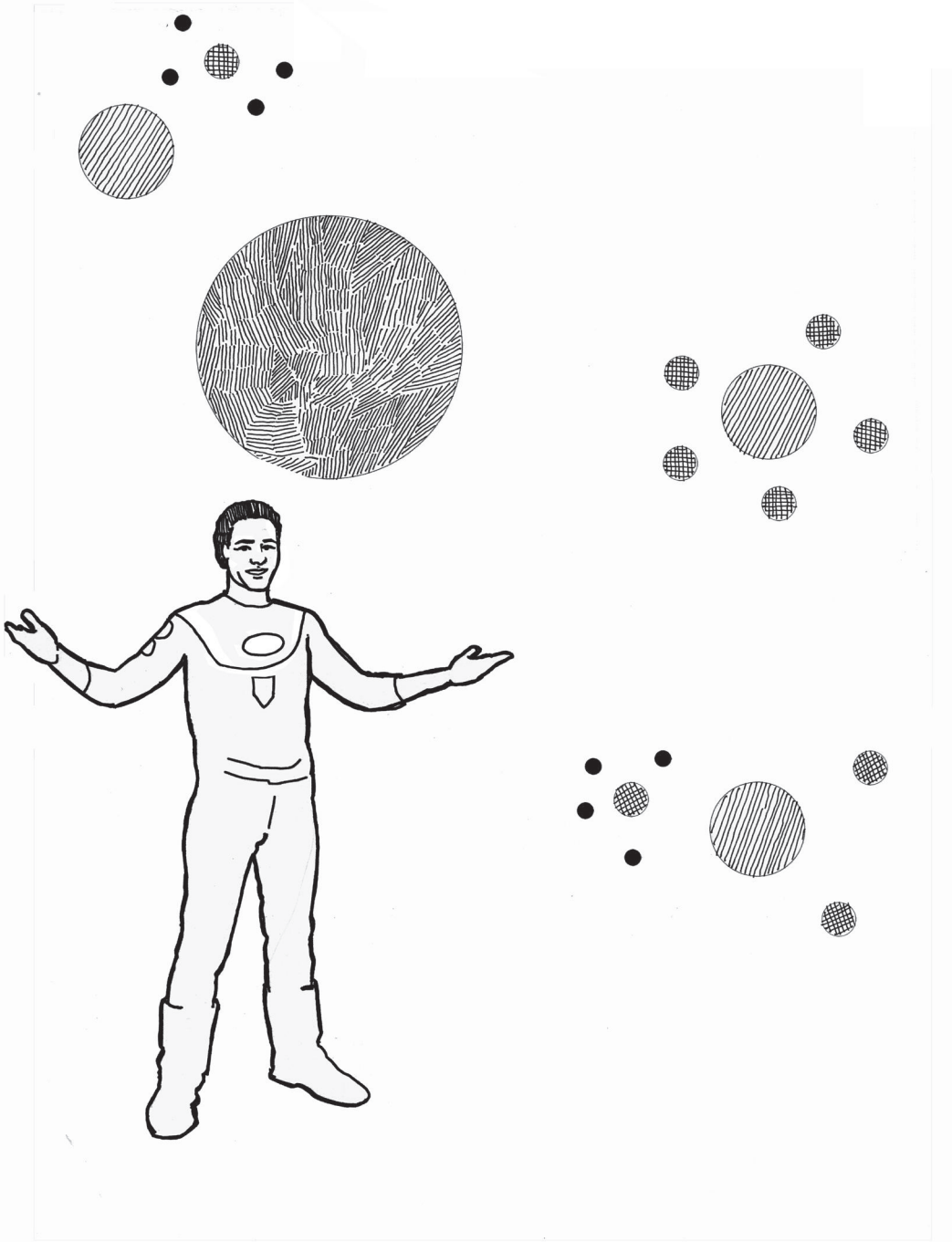
Kanban quindi ha a che fare con il miglioramento e con il cambiamento del modo di organizzare il lavoro. I suoi principi sono molto semplici.

Ci fu un momento in cui si misero in discussione i complessi strumenti e processi di project management, preferendo cose più semplici, tipo una lavagna su cui si attaccavano dei cartellini adesivi.

Sì, in effetti funzionò. Soprattutto perché questi semplici strumenti permettevano di instaurare la discussione tutti insieme di fronte a grafici o numeri. La condivisione delle informazioni contenute in questi "information radiator" era la cosa più importante.

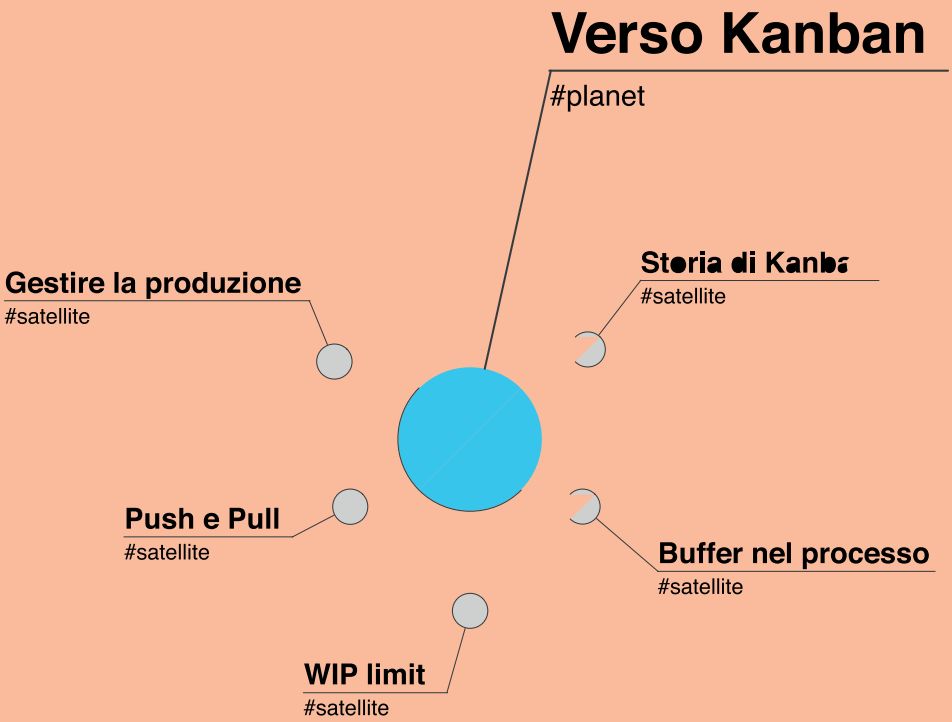

Interessante… project management fatto coi biglietti di carta…
E gli altri principi di Kanban quali sono?


Sono questi…
condividere un percorso di cambiamento che abbia come obiettivo la crescita e l'evoluzione dell'organizzazione
usare gli strumenti di cui si è già in possesso: partire da quelloche si sa fare senza aspettare di avere imparato nuove cose
rispettare l'organizzazione attuale



Su questo sistema solare vedremo quindi come, grazie all'uso di lavagne con bigliettini attaccati e un po' di matematica, sia possibile non solamente controllare un processo, ma anche introdurre nell'organizzazione un miglioramento a livello sistemico.

Capitolo 1
Origine e principi di Kanban



Introduzione ai principi Lean Agile

In questo capitolo introduciamo i concetti fondamentali della **metodologia Kanban** (con la "K" maiuscola); tramite una serie di esempi vedremo anche come implementare una **kanban board** (con la "k" minuscola).

NOTA: Esiste una differenza importante tra *Kanban* con l'iniziale maiuscola (la metodologia sviluppata da D.J. Anderson) e *kanban* con l'iniziale minuscola (il cartellino segnalatore utilizzato inizialmente nel Toyota Production System). In tal senso le parole di D.J. Anderson sono particolarmente esaustive: "...Kanban (con la 'K' maiuscola) è il metodo di cambiamento evolutivo che utilizza un sistema *pull* basato sui cartellini e sulle *board* kanban (con la 'k' minuscola), sulla visualizzazione dei flussi e su altri strumenti per catalizzare l'introduzione di idee Lean all'interno del mondo dello sviluppo tecnologico e delle attività IT".

Vedremo come, tramite Kanban, sia possibile gestire un processo di produzione: in tal senso Kanban può essere considerata anche una metodologia di *project management*. Ma vedremo soprattutto come sia possibile osservare, valutare, far emergere i problemi per poi risolverli. Per questo motivo personalmente considero **Kanban** un vero e proprio strumento di **sense making**.

I principi base sui quali si poggia la metodologia sono per la maggior parte tratti dal lavoro sviluppato in Toyota da Taiichi Ōno durante la formulazione del cosiddetto **Toyota Production System**, noto nel mondo occidentale come **Lean Production**. Per questo motivo, per comprendere **Kanban** è impossibile prescindere dalla comprensione dei principi base della **Lean** tanto le due cose sono strettamente collegate fra loro.

È per questa ragione che ho pensato di cominciare questo capitolo presentando una serie di esempi, presi dalla vita di tutti i giorni, con cui spiegare in modo semplice concetti come **just in time**, produzione **trainata** dal mercato, dimensione dei **lotti** di produzione, limitazione delle attività **in parallelo**; tutti aspetti alla base della **Lean Production** e che tramite **Kanban** si possono tenere sotto controllo per migliorare il processo di produzione.

Non entreremo nel dettaglio della produzione snella e del Toyota Production System, sebbene una comprensione dei principi di base sia utile se non indispensabile. Per chi fosse interessato a maggiori approfondimenti su questi argomenti, si consiglia la lettura dei due testi più importanti in materia: *Toyota Way* [TW] e *La macchina che ha cambiato il mondo* [MCW].

La pizza al taglio, ossia il Lean nel negozio all'angolo

Il primo esempio che presentiamo è quello di una pizzeria al taglio che ha aperto da poco in paese. In questa piccola bottega c'è un pizzaiolo esperto che, dopo molti anni di lavoro alle dipendenze di altri principali, ha deciso di mettersi in proprio. Useremo un nome di fantasia per questo simpatico ristoratore e nella migliore tradizione della pizza napoletana, lo chiameremo Antonio, per gli amici "Totò".

Antonio, nel piccolo negozio ha messo un **banco** con una **vetrina** dove gli avventori possono vedere le **pizze** appena sfornate, già tagliate in **fette** o **spicchi**. Accanto alla vetrina c'è il bancone per le consegne e una piccola cassa.

Normalmente Antonio tiene in questa vetrina quattro o cinque tipi di pizza, scelte fra quelle più richieste dalla clientela. Il pomeriggio, all'uscita della scuola vicina, molti bambini si fermano coi genitori per fare merenda e per questo motivo una delle pizze più richieste è una specie di focaccia dolce farcita di cioccolata. La sera invece gli adolescenti preferiscono pizze dai sapori più forti, piccanti o magari qualche ricetta più dietetica.

Dietro il bancone Antonio ha installato una macchina **spianatrice** — con la quale, partendo da palle di pasta preparata manualmente, si formano le basi di pasta cruda — e un paio di **forni** elettrici per la cottura.

Antonio, da esperto pizzaiolo, è in grado di sfornare sempre la giusta quantità di pizze per ogni tipologia, tanto che normalmente l'attesa media dei clienti è sempre molto ridotta.

Gestire una pizzeria con un approccio Lean

Il modo con cui Antonio gestisce la sua pizzeria è molto efficiente e vi si possono trovare applicati molti dei principi della filosofia di produzione Lean messa a punto da Toyota nel dopoguerra: gestione del magazzino **just in time**, produzione **pull** "trainata"

Figura IV.1. Antonio, il pizzaiolo della pizzeria a taglio sotto casa, applica i principi della Lean Production che sono i capisaldi della metodologia Kanban.

dal mercato, limitazione del **work in progress** sono tutti principi utili quando si deve gestire un processo di produzione quale esso sia: dalla multinazionale alla pizzeria sotto casa.

Produzione snella: approccio pull e gestione del magazzino just in time

Nella pizzeria, Antonio fa in modo di avere, in ogni momento, **poche pizze pronte** nella sua vetrina, cosicché la clientela possa sempre gustare pizze calde perché appena sfornate.

Lo scopo di Antonio è quello di ottimizzare la produzione, ossia fare in modo che il bancone non sia mai completamente vuoto — i clienti dovrebbero aspettare tutto — o troppo pieno — pizze fredde da riscaldare — ottenendo così anche un altro risultato: a fine giornata, gli sprechi saranno stati minimi.

In termini Lean, potremmo dire che Antonio applica un processo di lavorazione "trainato" dal mercato (approccio **pull**) e organizza la sua produzione secondo un meccanismo **just-in-time**, per cui non si creano in anticipo grandi sovrapproduzioni che poi devono essere vendute per svuotare il bancone, corrispondente in termini industriali al magazzino dei prodotti finiti. Entrambe queste strategie sono possibili grazie al fatto che nel processo di lavorazione sono inserite delle "aree di parcheggio", di cui parleremo diffusamente sotto.

Nei momenti in cui non c'è molto da fare, quando per esempio il bancone è pieno o c'è poca clientela nel negozio, Antonio si mette a svolgere quelle attività non urgenti, che potremmo definire **a bassa priorità**, come per esempio preparare le palle di pasta che poi andranno in frigorifero, oppure pulire il laboratorio, oppure riposarsi.

Il suo obiettivo è fare in modo che sul bancone ci siano sempre tre o quattro pizze pronte: conoscendo il tempo necessario per realizzare una pizza (il cosiddetto **cycle time di processo**), appena vede che le scorte scendono sotto un certo quantitativo, si mette a preparare una nuova pizza.

È quindi il flusso dei clienti che comprano le pizze, ossia il **ritmo** di **svuotamento** del **magazzino**, che determina il processo di lavorazione: in gergo si dice che il **processo viene tirato** (*pull* in inglese) dal fondo.

"Spingere" la produzione: approccio push

Se Antonio applicasse un approccio opposto (**push**), si concentrerebbe sulla produzione delle pizze cercando prima di tutto di immaginare quante ne riuscirebbe a vendere: in questo caso la sua unica preoccupazione sarebbe quella di "spingere" la produzione dal forno verso il bancone e quindi verso la bocca dei clienti, in modo da non accumulare troppe pizze pronte per essere infornate. Per perseguire questo approccio, dovrebbe fare molta attenzione alla pianificazione e controllare accuratamente l'organizzazione del suo processo di lavorazione, valutando in particolar modo i vari "colli di bottiglia" della produzione: il forno, la preparazione delle palle di pasta, il bancone e così via.

Antonio sa che questo lavoro di controllo e pianificazione è indubbiamente più complicato, sicuramente meno efficiente dell'altro (**pull**); inoltre a fine giornata qualche spreco in più lo si avrebbe sempre.

Messa a punto delle varie attività: task lenti, task bloccanti, task poco importanti

Antonio nella sua piccola bottega mette in atto anche altri **principi** tipici della **produzione snella**. Egli infatti lavora da **solo**, anche se in quel negozio ci sono un **paio** di forni e una sola pressa che spiana le palle di pasta per la pizza.

Questa configurazione non è casuale. La macchina pressatrice infatti svolge un compito molto **semplice** e piuttosto **rapido** che dura solo pochi secondi. Il forno invece è probabilmente la fase della lavorazione più **lunga**, dato che per cuocere una pizza sono necessari circa alcuni minuti a seconda del tipo di pizza. Per questo motivo, nella stanza **una pressatrice** è più che sufficiente mentre disporre di **un solo forno** avrebbe rallentato il processo, creando code in attesa. In realtà dopo qualche tempo Antonio si è potuto permettere un forno più grande, in grado di cuocere fino a dieci pizze contemporaneamente, cosa che, di fatto, permette non solo di velocizzare il tempo di preparazione di una singola pizza, ma di eliminare ingolfamenti in entrata al forno. In questo modo egli può smettere di preoccuparsi della gestione delle varie fasi intermedie e concentrarsi sull'unico punto veramente importante: il numero di pizze pronte ed esposte nella vetrina.

Piccoli lotti di lavorazione (one piece flow)

Una cosa che Antonio gestisce con la massima attenzione è la fase più delicata della lavorazione, la **cottura**: essendo la fase più lunga, e la più delicata, dell'intero processo, è molto importante non trascurarla. Per questo motivo egli spesso sacrifica altre fasi della lavorazione: sia che stia guarnendo una base di pasta, sia che stia servendo un cliente, spesso si interrompe per infornare o per togliere una pizza pronta (**task lento** ad **alta priorità** perché **bloccante** sugli altri).

In realtà Antonio gestisce quest'alternanza, per esempio fra il servire un cliente o interrompersi per infornare una pizza, in modo estremamente attento; come ogni commesso che lavora al pubblico, conosce bene il valore di ogni singola fase del suo lavoro: per esempio interrompere il condimento di una pizza ha un impatto diretto solo sul processo di lavorazione. Sospendere momentaneamente il servizio di un **cliente** è invece una cosa che potrebbe costare un prezzo maggiore in termini di immagine o di soddisfazione della clientela.

Dato che sa intrattenere con simpatia i propri clienti, può permettersi di interrompere il servizio al bancone per cuocere una pizza; Antonio sa che però può farlo solo per pochi istanti. Se per esempio decidesse di seguire la lavorazione di cinque pizze per volta, tutte le fasi sarebbero più impegnative in termini di tempo e quindi anche le interruzioni al servizio al bancone diventerebbero intollerabili per la clientela.

Oltre al problema della gestione della clientela, l'esperienza ha ormai insegnato ad Antonio, che preparare più pizze contemporaneamente non è conveniente: a volte si creano "ingorghi" durante la lavorazione delle pizze, per esempio in entrata del forno, oppure ci sono fasi della lavorazione completamente scariche (p.e., vetrina vuota). Antonio sa quindi che il trucco è gestire **lotti di lavorazione piccoli**, per esempio una o due pizze per volta, limitando il numero di attività che svolge in parallelo. In questo modo, nel tempo ha scoperto che la produzione si velocizza e la clientela è più contenta. Senza saperlo, Antonio, limitando il numero di operazioni svolte in parallelo e gestendo quindi lotti piccoli, sta applicando uno dei principi fondamentali della **produzione snella** messa a punto in Toyota.

La legge di Little

Infatti, mella definizione del Toyota Production System, grazie al lavoro di Taiichi Ōno si comprese l'importanza di **velocizzare il flusso** della lavorazione riducendo il più possibile i lotti di produzione. Gli studi sulla teoria delle code e più precisamente della "legge di Little" confermano la bontà di queste idee.

John D.C. Little è un fisico statunitense noto fra le altre cose per aver formulato la legge che da lui prende il nome: "il numero medio di clienti in un sistema è uguale al tasso medio di arrivo moltiplicato per il tempo medio nel sistema".

Riportando il discorso nell'ambito di un processo produttivo si potrebbe dire che il numero di **pezzi in lavorazione** (**WIP**, *work in progress*) è dato dal **tempo di attraversamento** (**Ta** , ossia il tempo è necessario per lavorare un pezzo) moltiplicato per un coefficiente di **prestazionalità** (**Th**, il *throughput* del sistema, ossia quanto si è veloci nel produrre un singolo pezzo). Questa relazione è esprimibile secondo l'equazione:

```
WIP = Th * Ta
```

Tornando all'esempio della pizzeria, normalmente un cliente che entra nella pizzeria di Antonio vorrebbe essere servito prima possibile; analogamente, ad Antonio interessa ridurre il tempo di lavorazione in modo che sia possibile limitare il **magazzino** (numero di pizze pronte) per evitare sprechi (pizze fredde o avanzi a fine giornata). Dato che la filiera è corta e non ci sono dipendenze da sistemi esterni, in questo caso **tempo di attesa** e **tempo di lavorazione** sono praticamente (quasi) la stessa cosa. Nei casi più complessi sarebbe necessario aggiungere un fattore correttivo, ma la sostanza non cambia: si deve trovare un modo per **ridurre** al minimo possibile il `Ta`. Ribaltando la formula di prima si ottiene che:

```
Ta = WIP / Th
```

ossia un `Ta` piccolo si può avere sia aumentando le prestazioni, sia **riducendo** il numero di cose che **contemporaneamente** sono **in lavorazione**.

Aumentare le prestazioni del sistema non sempre è cosa facile, mentre diminuire la dimensione del lotto è certamente realizzabile. Ridurre al minimo il lotto di lavorazione porta a quello che nel sistema Toyota viene chiamato **one piece flow**: si produce **un pezzo per volta** sulla base delle effettive richieste che arrivano dal campo.

Lo *one piece flow* spesso non è realizzabile a causa della variabilità del sistema (variazioni della domanda, dipendenze da linee di lavorazioni esterne, setup operativi); Per questo spesso si introducono delle aree di sosta: sono i cosiddetti **buffer**, elementi in grado di assorbire tale variabilità.

Le aree di sosta nel processo: i buffer e la gestione delle code

Gran parte delle buone pratiche che Antonio riesce a mettere in atto nel suo negozio (**pull**, **just in time**, **WIP limit**) sono possibili grazie a un altro importante elemento: le zone di stazionamento rappresentate nel nostro esempio dal piano di lavoro dove prepara e appoggia i semilavorati, o dalla vetrina delle pizze pronte.

Nella terminologia **Lean** queste aree sono dette **buffer di processo**, ossia vere e proprie "aree di parcheggio" in grado di assorbire gli sbalzi del processo di lavorazione o di domanda. Il **mura**, che in giapponese significa appunto "sbilanciamento", in Lean è una delle forme di spreco più gravi.

Grazie all'uso di tali aree di sosta, si può realizzare un processo in modalità **pull**: quando termina la lavorazione di un **semilavorato** di prodotto — nel nostro esempio potrebbe essere la pizza stesa ma non ancora farcita — questo viene appoggiato nella zona di parcheggio — il piano di lavoro vicino al forno — in attesa che sia preso ("tirato", *pull*) e messo in lavorazione per la fase successiva.

Inoltre, per mezzo dei **buffer** è possibile gestire eventuali dipendenze da fasi esterne del processo. Per esempio, una pizza con prosciutto crudo deve essere guarnita dopo la cottura: al termine del tempo di cottura, Antonio può estrarre la pizza dal forno, appoggiarla su un piano e guarnirla con calma per esempio dopo aver infornato altre pizze. Se lasciasse la pizza in forno si brucerebbe, se si mettesse a guarnirla prima di infornarne altre, sprecherebbe del tempo prezioso, dato che la cottura è un processo lento e autonomo.

La **dimensione** di tali **buffer** — per esempio il numero di pizze sul piano prima del forno, o delle pizze cotte ed esposte in vetrina — è una variabile molto importante: un **buffer** troppo piccolo (poche pizze in vetrina pronte per essere vendute) rischia di esaurirsi prima che Antonio abbia il tempo di preparare altre pizze; se il buffer invece è troppo grande, il pericolo è quello di non riuscire a vendere tutte le pizze esposte prima che queste perdano la loro freschezza e si rovinino. Normalmente, sia per l'esempio delle pizze che nei processi di produzione più complicati, il valore ottimale si ottiene dopo alcuni **esperimenti** sul campo; ma torneremo su questo aspetto quando parleremo di progettazione della **kanban board**.

Analogamente decidere il numero e la posizione dei buffer nella catena di lavorazione non è banale: in questo caso, essendo la lavorazione è a **catena corta**, possono bastare

due soli buffer lungo la linea di lavorazione per stabilizzare il processo. Nei casi più complessi, i processi di produzione si avvalgono di uno o più **stazioni di parcheggio**.

In ottica di miglioramento del processo, informazioni utili si possono ricavare dal conteggio del **numero** di **attività parcheggiate o dalla misurazione** della loro **permanenza nelle aree di parcheggio**. In linea di principio, buffer troppo affollati o attività parcheggiate da troppo tempo, sono sintomi di un sistema sub performante.

Kanban quotidiano: dal giardino imperiale alle code in autostrada

Per iniziare a parlare di Kanban in modo più specifico, possiamo prendere in esame uno scenario completamente diverso, ma molto esplicativo, che probabilmente ogni lettore avrà vissuto in prima persona nella sua vita di tutti i giorni: lasciare l'auto in un parcheggio a pagamento in cui **entrata** e **uscita** siano **regolati** da sbarre automatiche, sistemi di pagamento e, soprattutto, **posti limitati** e **numerati**.

Se il numero di automobilisti che desiderano parcheggiare è ben al di sotto della capienza del parcheggio, in genere nessuno si accorge del meccanismo che regola il parcheggio stesso: si arriva, si preleva un biglietto alla macchinetta, la sbarra si alza e si procede verso un posto libero.

Se invece il parcheggio ha raggiunto la **capacità massima**, normalmente all'entrata si formano delle **code** di auto in attesa di poter entrare nel parcheggio. In questo caso, essendo il numero di posti disponibili limitato dalla capienza massima del parcheggio, per **ogni** automobilista che **lascia** il parcheggio si **libera** un **posto** per la vettura di un altro che vuole entrare.

Questa organizzazione è possibile perché un semplice **contatore** tiene traccia delle auto che entrano e di quelle che escono e blocca il rilascio di nuovi biglietti quando si arriva a un certo limite.

Tale sistema si basa su un meccanismo inventato molti anni fa presso il parco del Palazzo Imperiale giapponese dove, in primavera, moltissime persone accorrono per poter assistere al bellissimo spettacolo della fioritura dei ciliegi; per garantire l'armonia e la bellezza del parco è necessario limitare il numero di visitatori contemporaneamente presenti nel giardino, cosa che fin dai tempi antichi viene ottenuta con un meccanismo estremamente semplice ma altrettanto efficace.

Pur essendo l'ingresso al parco gratuito, all'entrata ogni visitatore deve prelevare da un contenitore un "gettone", una **tessera visuale**: questo è il significato del termine giapponese **kanban** (scritto con la "k" minuscola) che vuol dire appunto, "cartellino", "tessera".

Nel giardino imperiale ogni visitatore deve tenere con sé questa **tessera** per tutto il tempo in cui rimane dentro il parco. Quando esce, dovrà riporre la sua **kanban** nel contenitore apposito. Quando non ci sono tessere **kanban** nel contenitore, nessun visitatore può più entrare nel parco: avere la tessera in mano è la condizione per poter entrare. Non appena un visitatore esce, depone il suo "gettone" kanban e un altro visitatore può riceverlo ed entrare a sua volta. Semplice. Lineare. Ripetibile.

In Toyota, fin dagli anni Cinquanta del secolo scorso, si è potuta tenere sotto controllo la produzione delle automobili in fabbrica proprio grazie a questo metodo di gestioe del flusso: contenitori e **kanban cards** (dei cartellini plastificati), senza l'ausilio di complessi sistemi informativi.

Le code in autostrada

Un'altra interessante esperienza che ogni lettore ha sperimentato almeno una volta è quella che si vive quando si percorrere in **automobile** un tratto di una **autostrada** particolarmente congestionato. In questo caso si può facilmente osservare che maggiore è il numero di auto che percorrono quel tratto, minore sarà la velocità con la quale si potrà percorrere quel tratto.

A volte tali rallentamenti portano a improvvisi e temporanei, quanto misteriosi, blocchi della circolazione del traffico, che poi magicamente riprende a muoversi senza un apparente motivo: non ci sono incidenti, non ci sono blocchi della viabilità. Questo fenomeno in gergo si chiama **Phantom Work Jam** o "ingorgo fantasma" (concetto approfondito al Capitolo 7 del libro di Stephen Denning [RM]).

Il **Phantom Work Jam**, oltre che essere legato al numero di auto che transitano per un determinato tratto di strada, è anche funzione della velocità di percorrenza media: **numero** di **auto** in **transito** e **velocità media** sono infatti due fattori che stressano il sistema in modo diretto.

Anche in questo caso il fenomeno è spiegabile applicando la legge di Little: essendo costante l'ampiezza delle corsie dell'autostrada (ossia il troughtput `Th`, in questo caso detto anche "capacità di canale"), maggiore è il numero di auto in transito (`WIP`), maggiore sarà il tempo di attraversamento (`Ta`), ossia si viaggia più lentamente. Più ci si avvicina al **limite massimo del sistema**, maggiore sarà la probabilità che si verifichino ingorghi.

Dato che un'autostrada può essere assimilata a un processo di produzione molto lungo (filiera lunghissima), nei momenti di criticità entra in gioco un altro fattore, il cosiddetto **effetto Forrester** (o "effetto frusta" o *bullwhip effect*) di cui si parla nell'Appendice A.

Per semplicità possiamo dire che quando siamo al limite della capacità del sistema, piccole oscillazioni si propagano amplificandosi lungo tutta la filiera. Un piccolo rallentamento di un'auto si trasmette aumentando il suo effetto, fino al punto in cui alcuni automobilisti sono costretti a una brusca frenata o al blocco del veicolo. Forrester ci dice poi che per smaltire l'ingorgo serve molto più tempo di quello che è necessario per la sua creazione.

Nel caso del traffico stradale, non essendo possibile nella maggior parte dei casi limitare il numero di auto in transito, un buon modo per evitare ingorghi fantasma è ridurre il limite di velocità; in tal senso alcuni interessanti esperimenti alla viabilità sono stati condotti in Gran Bretagna.

Breve storia di Kanban: dai cartellini al Project Management

Verso l'inizio degli anni Quaranta del secolo scorso, gli ingegneri di Toyota stavano lavorando all'ottimizzazione dei propri processi di produzione. **Taiichi Ōno**, anche prendendo spunto dal lavoro di William Edwards Deming, si impegnò per introdurre nuovi principi di quella che sarebbe diventata poi la **produzione snella**: ne abbiamo parlato diffusamente nella prima Parte di questo libro, *Verso Agile*.

Una delle fonti che dette maggior ispirazione a Ōno arrivò da un settore inaspettato, quello della grande distribuzione alimentare. In quel periodo stavano nascendo i primi supermercati dentro i quali i commessi e gli addetti al rifornimento del magazzino dovevano trovare soluzioni ottimali proprio per gestire merci con un così alto tasso di degradazione: di sicuro, un cesto di insalata impiega molto meno tempo per perdere di "freschezza" rispetto a quello necessario a un modello di automobile per diventare obsoleto...

In quel caso, il flusso di approvvigionamento era guidato dal livello di rifornimento a scaffale: **solo** quando l'**ultimo elemento** di un certo prodotto era prossimo alla vendita, si procedeva al riordino; non prima. Furono quelli i primi esempi di gestione **just-in-time**, che ispirò i manager giapponesi della Toyota nel mettere a punto un nuovo processo di approvvigionamento.

Taiichi Ōno fu impressionato anche dall'elevato livello di efficienza di un supermercato in termini di risorse impiegate: in genere sono necessari solo pochi commessi, grazie al **pull** (i clienti svuotano gli scaffali con la merce) e al **just in time**.

Portando in Toyota le idee di Deming e tramite la **gestione** in tempo reale dei **flussi** di **entrata** e di **uscita** dei prodotti, nella fabbrica Toyota riuscirono a migliorare le prestazioni complessive del sistema: stava nascendo un nuovo modo di lavoro. Per chi fosse interessato a leggere tutta la storia della nascita del **Lean** e del **Toyota Production System**, si consiglia la lettura dei due testi più importanti [TW] e [MCW].

Segnale visivo

Il principio che animò questo processo di trasformazione è descrivibile con un semplice slogan: "migliorare la comunicazione tramite una gestione visuale del processo".

La parola **kanban** indica infatti una "segnalazione visiva", un "cartello" e infatti buona parte della metodologia, quando fu creata, faceva uso di **cartellini** visuali e di **lavagne** dove questi cartellini erano attaccati per monitorare gli **stati** di **avanzamento** della produzione, l'**approvvigionamento**, l'acquisto o la movimentazione dei **materiali**.

Il sistema, in modo estremamente naturale, stimolava il confronto e il dialogo all'interno del team, velocizzando lo scambio delle informazioni e agevolando quindi il processo decisionale sul cosa e come organizzare il lavoro. Fra gli obiettivi che Ōno e il suo staff si dettero durante il lavoro di messa a punto del Toyota Production System, vi era quello di velocizzare il flusso della lavorazione, limitare l'accumulo delle merci in entrata o dei prodotti finiti in uscita, di stabilizzare il flusso massimizzando le performance.

Kanban come metodologia di gestione e controllo

A partire dal 2007, **Kanban** (con la "K" maiuscola per distinguerlo dal cartellino) è stato introdotto come **metodologia** di **gestione** e **controllo** dello **sviluppo** del **software** grazie al lavoro di David J. Anderson, il quale si è ispirato ai principi del Lean e del Toyota Production System messo a punto da Ōno in oltre venti anni di lavoro. Anderson ha formalizzato le sue idee in un libro [SEC] pubblicato nel 2010.

Caratteristiche di Kanban

Nel suo lavoro, Anderson ha analizzato i principi della produzione snella e, sulla base di esperienze personali e di confronti con altre metodologie di gestione, ha tentato di adattare i principi della **produzione snella** alla **gestione** dello **sviluppo software**, integrando le sue idee nel più ampio quadro delle metodologie agili.

I cinque aspetti fondamentali

Nel libro, Anderson ha sintetizzato in **cinque punti** gli aspetti più importanti di Kanban:

- Visualizzare il flusso di lavoro: dare una rappresentazione visiva del processo di lavorazione, ponendo in risalto, per ogni step del processo, il valore prodotto.
- Limitare il Work-in-Progress (WIP): porre dei limiti espliciti sul lavoro eseguibile all'interno di ogni fase della lavorazione.
- Misurare e gestire il flusso: misurare e gestire il flusso per avere sempre informazioni attendibili e poter quindi prendere decisioni coerenti col sistema; per ogni azione intrapresa, visualizzare sempre le conseguenze.
- Esplicitare le regole del processo: è di fondamentale importanza la condivisione e il rispetto delle regole del processo da parte di tutti gli interessati; secondo Anderson questo è il modo più semplice ed efficace per ottenere gli obiettivi di processo, pur garantendo il miglioramento delle prestazioni e la riduzione degli sprechi.
- Identificare le possibili opportunità di miglioramento: creare nell'organizzazione una cultura in cui il continuo miglioramento del sistema, del processo e delle performance siano obbiettivi condivisi in tutti i membri della comunità; in giapponese, tale cultura del miglioramento continuo è nota con il nome di kaizen, parola ricorrente nel lavoro di Taiichi Ōno durante la creazione del Toyota Production System.

Tre regole importanti

Questi cinque punti possono essere considerati i cinque obiettivi fondamentali che dovrebbero essere perseguiti tramite **tre regole importanti**, che sono alla base di Kanban:

- Usare gli strumenti di cui si è già in possesso: partire da quello che si sa fare senza aspettare di avere imparato cose nuove.
- Condividere un percorso di cambiamento che abbia come obiettivo la crescita e l'evoluzione dell'organizzazione.

- Rispettare l'organizzazione attuale: il processo di cambiamento ed evoluzione non deve imporre stravolgimenti drastici nel processo, modifiche traumatiche nei ruoli e nelle persone, ma introdurre le novità in modo graduale e compatibile con le necessità e gli obiettivi. Per questo sono così importanti la visualizzazione del processo, la misurazione, la definizione esplicita delle policy.

Molte delle indicazioni fin qui viste sono prese per la maggior parte dai principi del **Lean** che **Kanban** sposa in pieno: si potrebbe quasi dire che l'obiettivo principale di **Kanban** è quello di funzionare come **attivatore** dei **principi Lean** all'interno dei sistemi di **produzione** del **software**. Taiichi Ōno era solito dire: "Lo scopo del kanban è di far emergere i problemi e metterli in relazione con l'attività di miglioramento continuo".

Conclusioni

Attraverso un certo numero di esempi relativi ad attività quotidiane e facilmente comprensibili, si sono introdotti i concetti essenziali di un sistema di produzione basato su principi Lean.

Abbiamo poi raccontato brevemente la storia dei cartellini kanban adottati inizialmente nel Toyota Production System e di come molti dei principi Lean utilizzati nella produzione industriale siano stati adattati al mondo della gestione dello sviluppo del software grazie al metodo Kanban, messo a punto da D. J. Anderson a partire dalla fine del decennio passato.

Nei prossimi capitoli affronteremo la metodologia Kanban sul piano operativo, illustrandone pratiche e strumenti.

Riferimenti

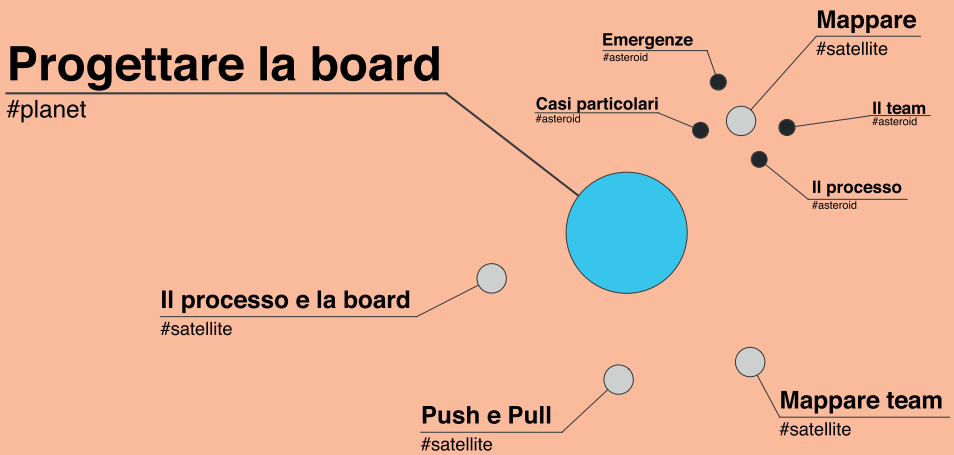
[TW] Toyota Way. 14 principi per la rinascita del sistema industriale italiano
http://www.toyota-way.it

[MCW] Womack J.P. – Jones D.T. – Roos D., *The Machine That Changed the World: The Story of Lean Production*, Harper Business, 1991 (trad. it. *La macchina che ha cambiato il mondo*, Rizzoli, 1999)

[SEC] David J. Anderson, *Kanban: Successful Evolutionary Change for Your Technology Business*, Blue Hole Press, 2010

[RM] Stephen Denning, *The Leader's Guide to Radical Management: Reinventing the Workplace for the 21st Century*, Jossey-Bass, 2010

Capitolo 2
Progettare e usare la kanban board



Kanban in pratica

In questo capitolo tramite una serie di esempi visuali, verrà mostrato come progettare e come utilizzare una **lavagna kanban**: lo scopo è quello di fornire delle indicazioni pratiche su come introdurre la metodologia Kanban all'interno della propria organizzazione.

Progettare la prima kanban board

Un modo per realizzare la lavagna kanban è quello di iniziare dalla raccolta su una **parete** o su un **pannello** di una serie di **cartellini** corrispondenti alle attività che compongono il processo che si vuol gestire con Kanban. L'uso di uno spazio ampio e libero da schemi predefiniti può aiutare a comprendere meglio lo stato attuale del lavoro all'interno dell'organizzazione (figura 2).

Successivamente si può procedere provando a schematizzare il **processo di lavorazione**, per esempio provando a raggruppare i cartellini corrispondenti ad attività simili. Un'altra strada potrebbe essere quella di ordinare i cartellini in base a una qualche regola: per esempio in alto quelli che dovranno essere presi in lavorazione per primi. Fra le molte possibilità, una pratica piuttosto comune è quella di iniziare la scomposizione della lavagna in colonne definendo tre aree: quella corrispondente alle attività ancora da svolgere, quelle in lavorazione e quelle già terminate (figura 3).

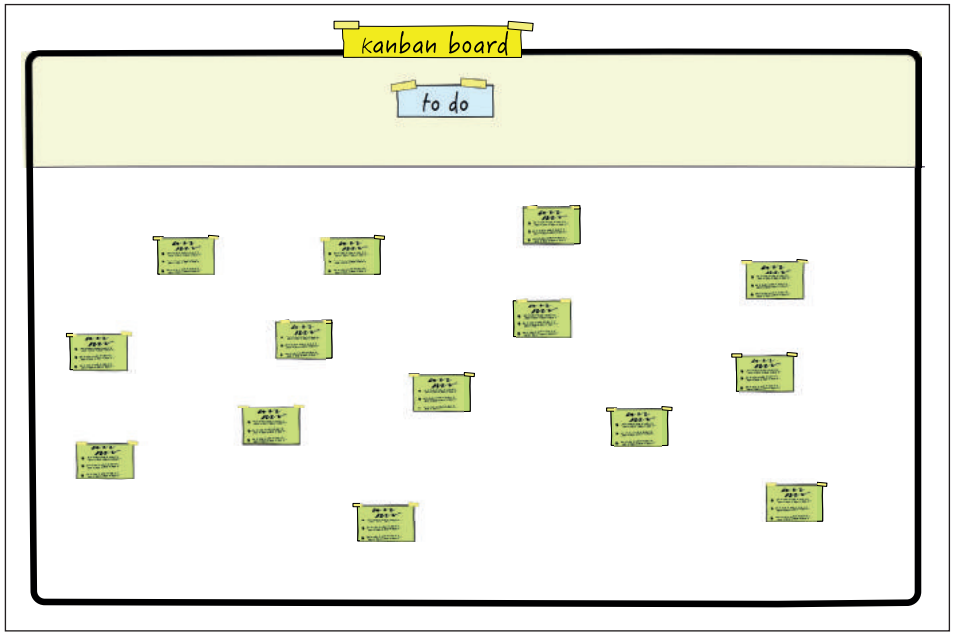


Figura IV.2. Per prima cosa si dispone in modo libero su una parete l'elenco delle attività che si dovranno svolgere.

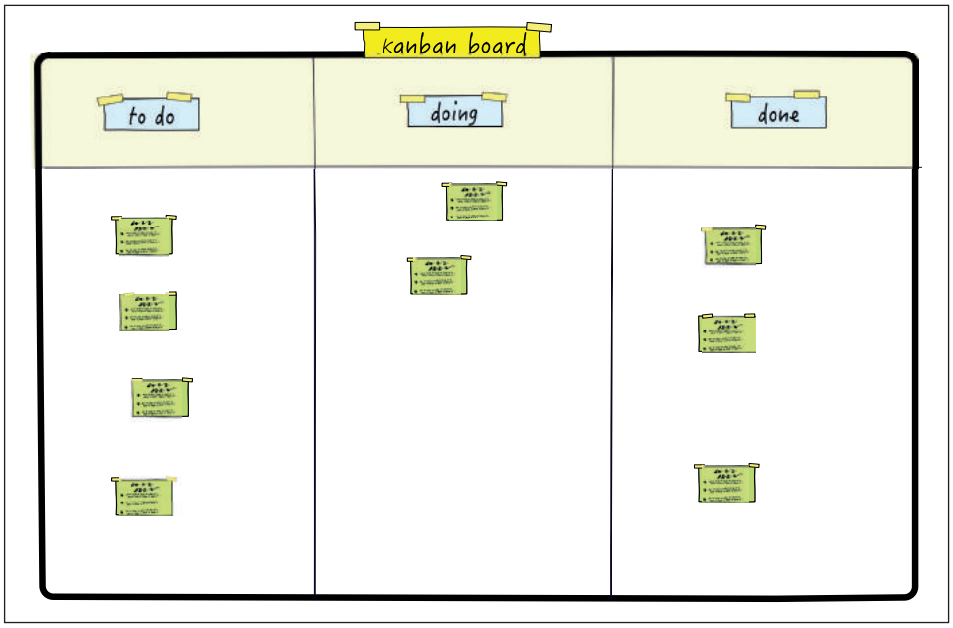


Figura IV.3. Il primo miglioramento che si può introdurre è una suddivisione in colonne per raggruppare le attività da svolgere, quelle in lavorazione e quelle che invece sono già terminate.

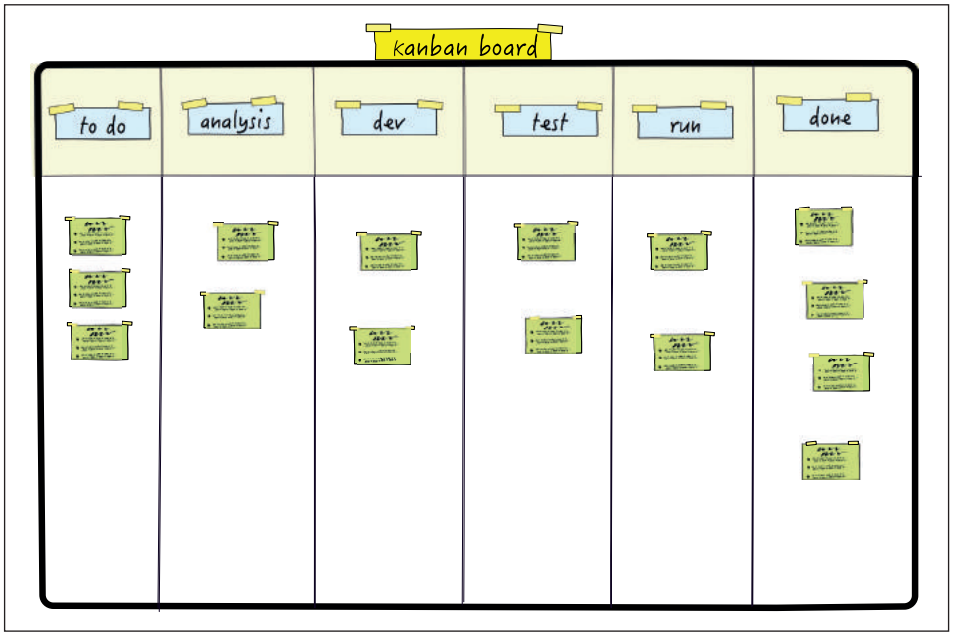


Figura IV.4. La colonna della lavorazione può essere scomposta in congrue attività di dettaglio. In un processo di sviluppo software, potrebbero essere le classiche analisi, sviluppo, test e deploy.

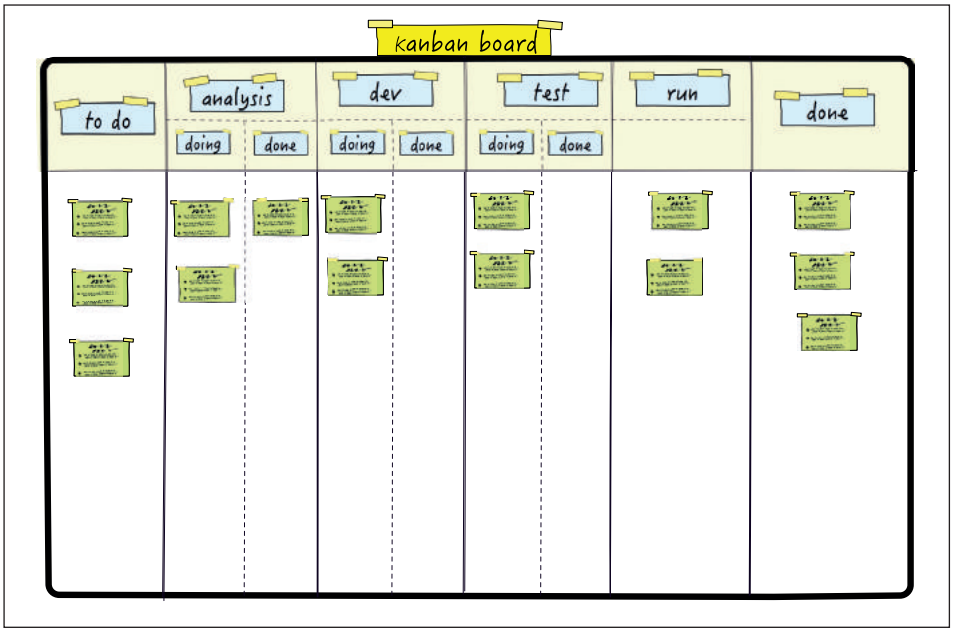


Figura IV.5. Il passo successivo potrebbe essere quello di introdurre le colonne di parcheggio, in modo da gestire il passaggio da un approccio push a uno pull.

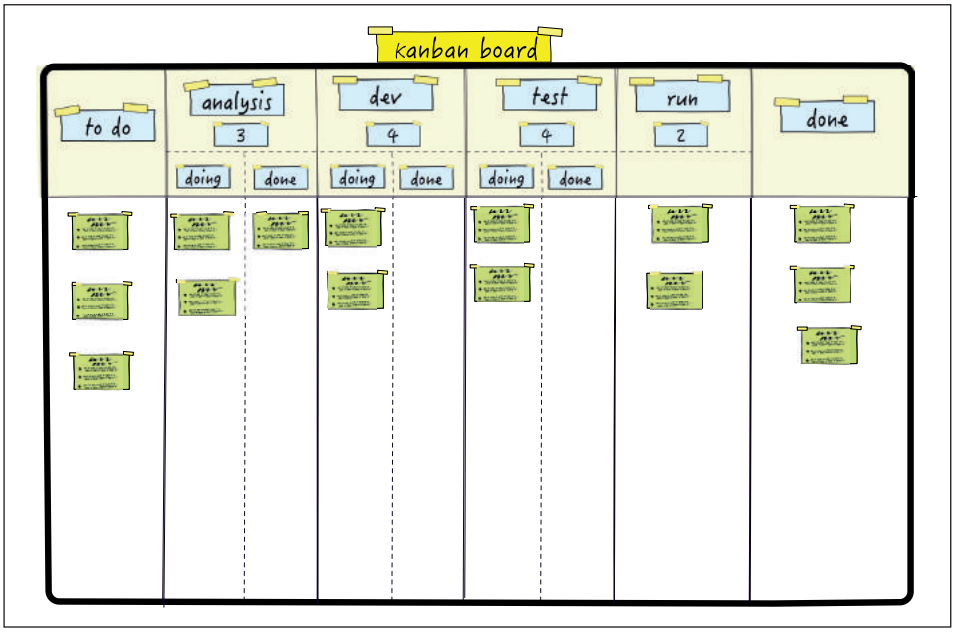


Figura IV.6. Come ultimo raffinamento si possono introdurre i limiti WIP (Work In Progress) sul numero massimo di elementi in lavorazione contemporanea.

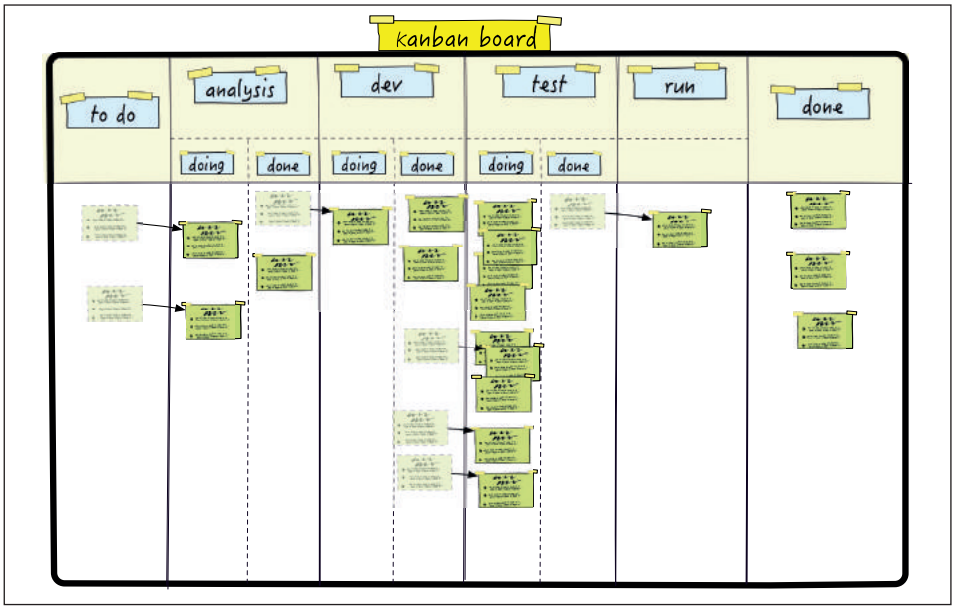


Figura IV.7. Le attività svolte in contemporanea (Work In Progress, WIP) non devono essere troppe. Se non si impone alcun limite di lavorazione (WIP limit), vi è una elevata probabilità che si verifichino degli ingorghi prima delle fasi lente (colli di bottiglia).

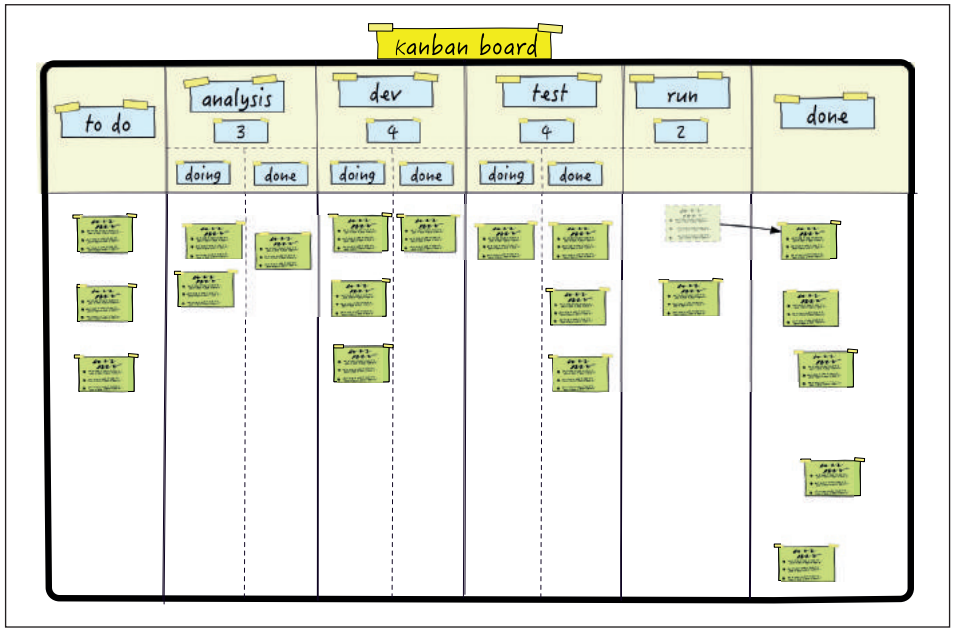


Figura IV.8. I limiti di lavorazione favoriscono l'approccio pull, evitando l'insorgere di colli di bottiglia delle attività.

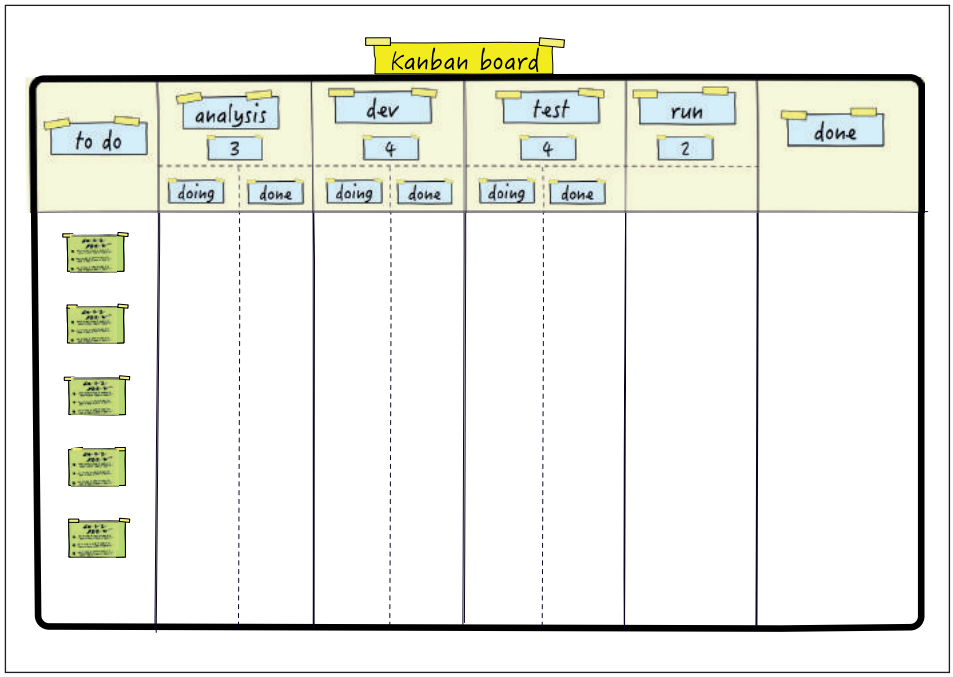


Figura IV.9. Situazione iniziale: ancora nessuna attività in lavorazione.

Successivamente per **dettagliare** ulteriormente il processo, si potrebbe **suddividere** la colonna corrispondente alla lavorazione, in modo da specificare le varie **fasi del processo**. Nella figura 4 per esempio sono state create le colonne corrispondenti alle tipiche fasi di un processo di sviluppo software: analisi, sviluppo, test e, infine, deploy (in questo caso identificato dall'etichetta **run**).

Nella figura 5 si nota l'introduzione delle colonne di **buffer** (o **code**) necessarie per "parcheggiare" le attività per le quali si è terminata la lavorazione relativa alla colonna corrente in attesa che siano messe in lavorazione nella fase seguente, il che comporta lo spostamento nella colonna successiva.

Infine, sulle varie fasi del ciclo di lavorazione si possono introdurre dei **limiti** al numero di attività (**WIP limit**) che sono eseguibili in contemporanea (figura 6).

In assenza di un **WIP limit** o in presenza di un valore non adatto, data la naturale propensione che molte persone hanno nell'iniziare nuove cose prima di aver terminato quelle in corso, si potrà notare un accumulo di cartellini in determinate posizioni sulla board, ossia dei veri e propri **ingorghi** di attività non completate

Nella board raffigurata nella figura 8 la presenza di un **vincolo imposto** sulle **colonne** favorisce — di fatto, obbliga — lo "scodamento" (approccio pull) dei cartellini sulle colonne di destra, impedendo quindi l'accumulo visto in precedenza in figura 7; in questo modo si ottiene una lavorazione più omogenea delle varie attività.

Simulazione di un flusso di lavorazione

Passiamo adesso a **simulare** la **gestione** di un tipico **processo** di lavorazione tramite una board kanban. Per fare questo si partirà con una situazione iniziale come quella raffigurata in figura 9, dove ancora nessuna attività è stata presa in carico per la lavorazione.

NOTA: il flusso che andremo a disegnare sulla board ricalca le tipiche fasi di un processo sequenziale che ricorda molto il modello a cascata (analisi, sviluppo, test e deploy). Questa scelta ha solo finalità di semplicità espositiva.

Il primo passo (figura 10) è quello di mettere **in lavorazione due attività** nella prima fase che è quella dell'analisi. Dato che il **WIP** (Work In Progress) **limit** della lavorazione è di **2 task**, questa operazione è consentita.

Non appena è **completata** la **lavorazione** di **uno** dei due task di analisi, questo verrà **parcheggiato** nella colonna delle **cose fatte** (figura 11), pronto per essere preso in lavorazione in sviluppo. In questo momento si libera un **WIP** sulla colonna di analisi per cui un altro task potrà essere messo in lavorazione in analisi.

Proseguendo la simulazione (figure 12 e 13) si osserva che i vari task si **spostano verso destra** dando luogo a un vero e proprio **flusso** di lavorazione.

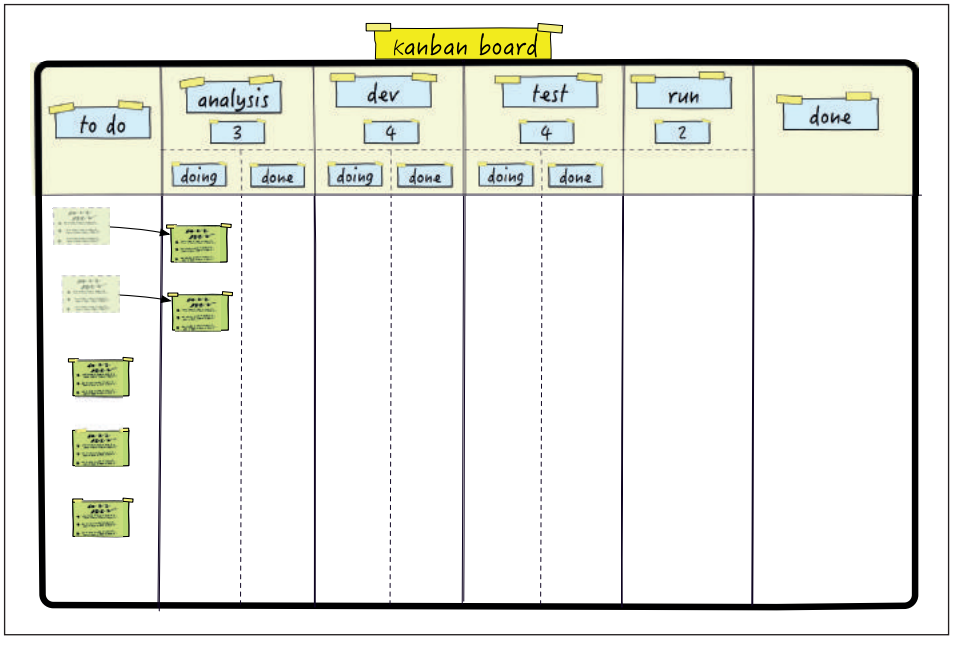


Figura IV.10. Si spostano verso destra le prime due attività dal backlog degli elementi pronti per la lavorazione.

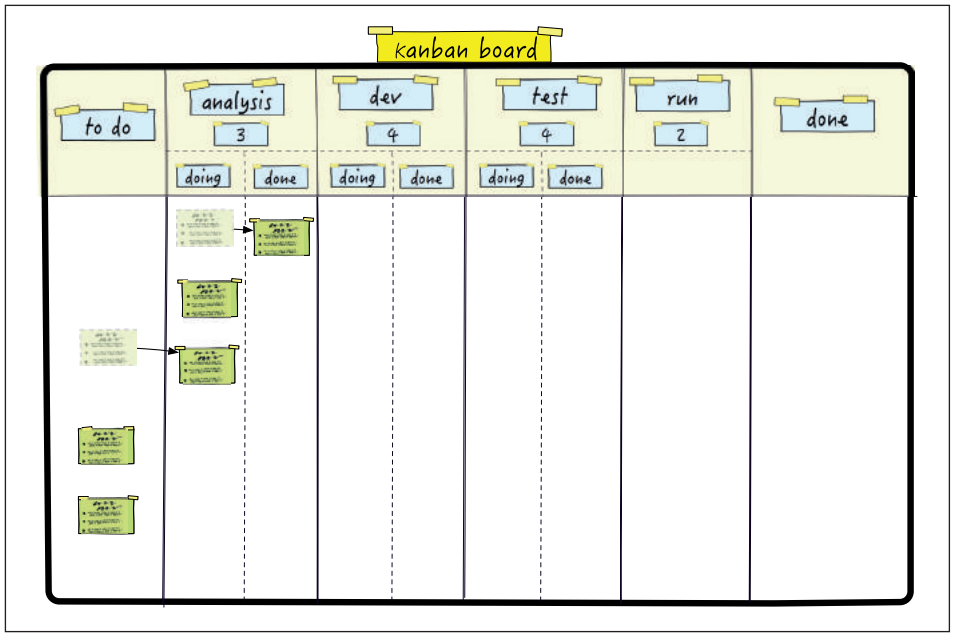


Figura IV.11. La prima attività ha completato la fase di analisi e viene spostata nella colonna del "done".

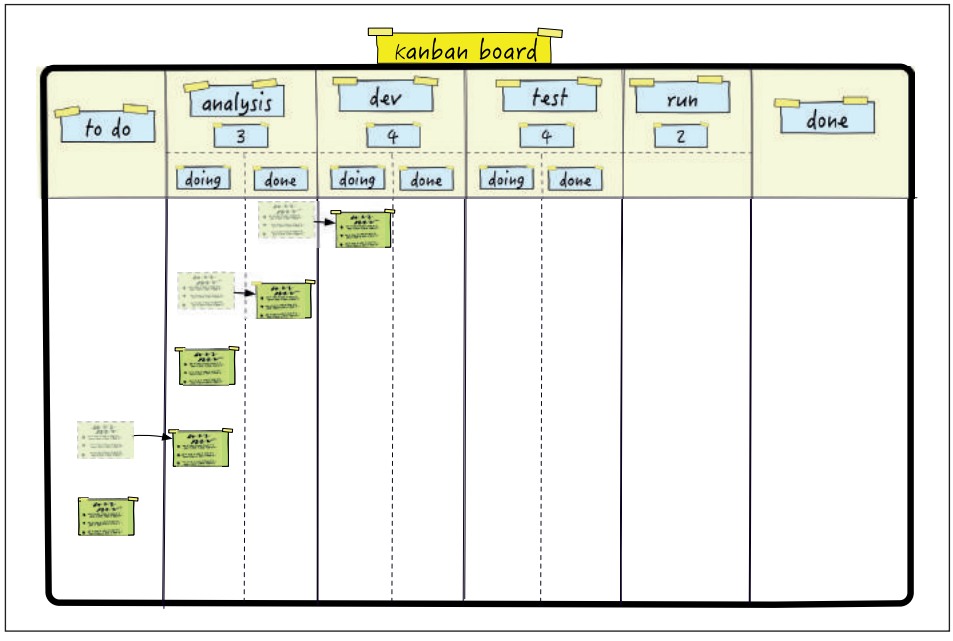


Figura IV.12. La prima attività viene spostata nella lavorazione "dev".

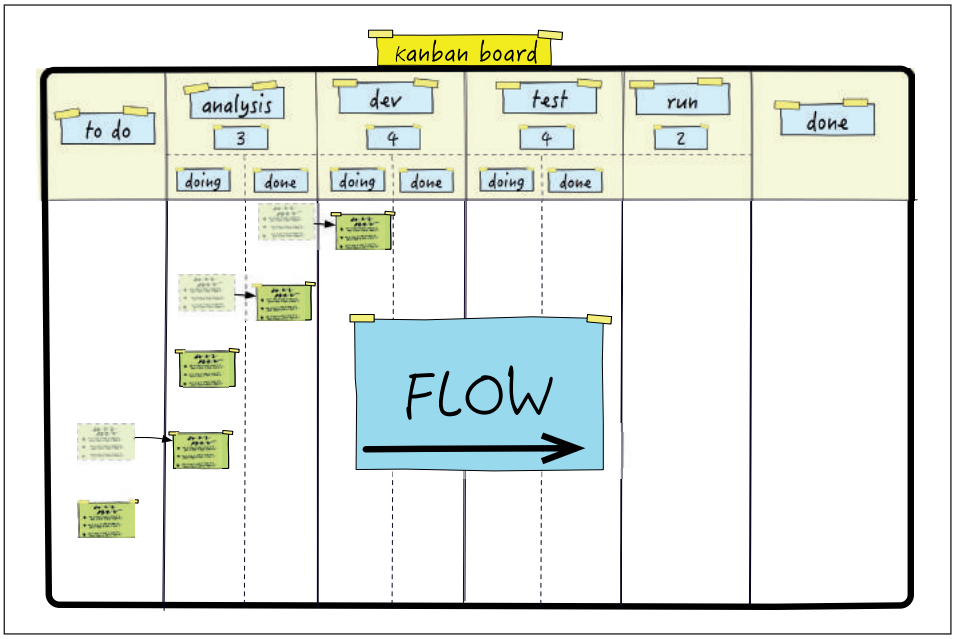


Figura IV.13. Con questo meccanismo si mette in atto un movimento verso destra che corrisponde al flusso della lavorazione.

Gli avatar del team

Spesso vi è la necessità di avere una visione aggiornata di **chi** sta svolgendo le attività in lavorazione: un modo semplice per ottenere questo obiettivo è quello di associare delle **icone rappresentanti** la persona o le persone che stanno effettuando la lavorazione.

Per le icone i possono utilizzare figure di fantasia, disegni astratti o vere e proprie foto delle persone (figura 14).

Quando una **persona** prende **in carico** un'attività, preleva il proprio avatar e lo **associa al cartellino**; viceversa, quando l'**attività** è **terminata**, l'**avatar** viene staccato dal cartellino e rimesso nell'area di **parcheggio**.

L'uso degli avatar permette di verificare in modo semplice e immediato l'impegno delle varie persone del team sulle varie attività, così come di coordinarne lo spostamento quando si presentano attività critiche che richiedano un rapido intervento.

Inoltre, limitando il numero degli avatar disponibili fisicamente sulla board (per esempio mettendone solo 3 o 4 per ogni persona), si può ridurre il numero di cose che ogni membro del team svolge in parallelo.

Se si guarda per esempio la figura 15, notiamo che Andrea è impegnato su tre differenti attività, quindi la sua capacità operativa è al limite massimo: dovrà terminare le attività che ha in carico prima di prenderne altre.

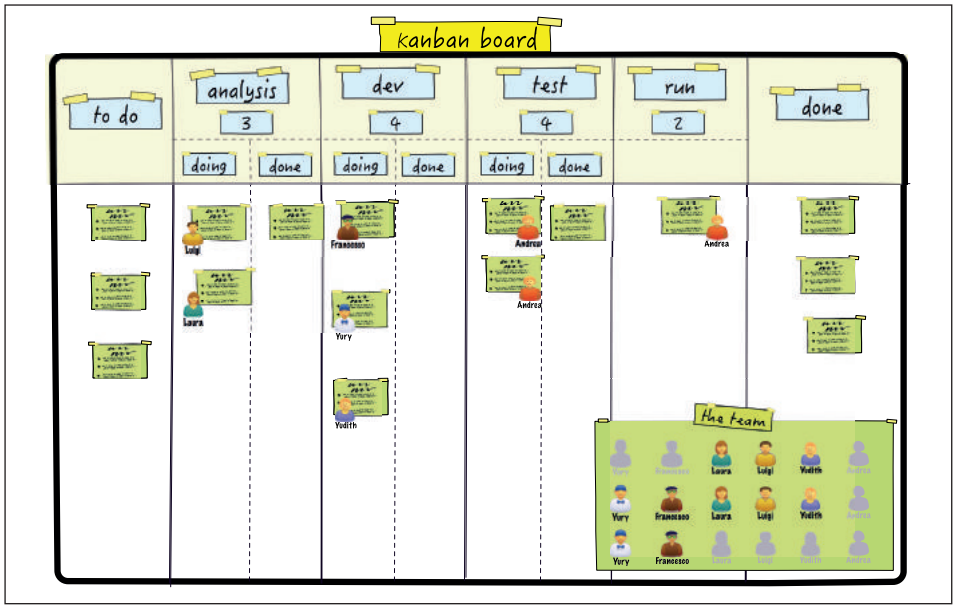


Figura IV.14. Gli avatar sono molto utili per rappresentare in modo rapido e visuale chi sta facendo cosa. In questo caso una parte della board è utilizzata come deposito di tutte le **icone** *delle* **persone** *del team (parcheggio degli avatar).*

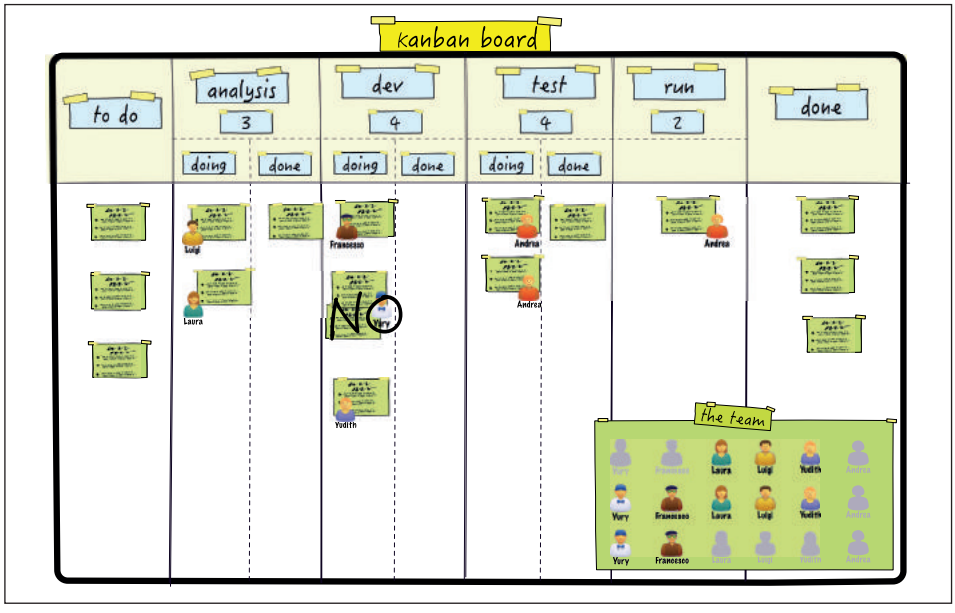


Figura IV.15. L'area di parcheggio può contenere un numero limitato di avatar per evitare che la singola persona faccia troppe cose in parallelo.

Il numero degli avatar da assegnare a ogni membro del team può essere uguale per tutti, oppure come in realtà accade, è funzione del tipo ti attività che le persone devono svolgere.

Il lavoro in un team cross-funzionale

Nelle figure 16, 17, 18 e 19 viene raffigurata una tipica situazione in cui le persone del team si spostano fra le varie attività o lavorano in coppia per velocizzarne il completamento. Questo tipo di organizzazione si verifica quando il team si dice **cross-funzionale**, ossia quando è composto da persone capaci di svolgere diverse attività previste dal processo, se non addirittura tutte.

Avere team cross-funzionali non deve essere visto come un obiettivo obbligatorio, ma offre certamente molti vantaggi; per esempio migliora l'efficienza, riduce le dipendenze dai singoli — se un membro del team è assente, si possono comunque portare a compimento le attività in carico a lui—, abbassa la pressione sul singolo distribuendo attività ai colleghi, previene i colli di bottiglia e altro ancora.

L'uso degli avatar sulla board permette al team di organizzarsi meglio e di far "fluire" le competenze in maniera adeguata.

Si può comprendere come questo sia possibile per esempio seguendo nella figura 16 e successive, gli spostamenti dell'avatar di Yury, il quale, dopo aver terminato un'attività di sviluppo, sposta il cartellino corrispondente nella colonna del finito.

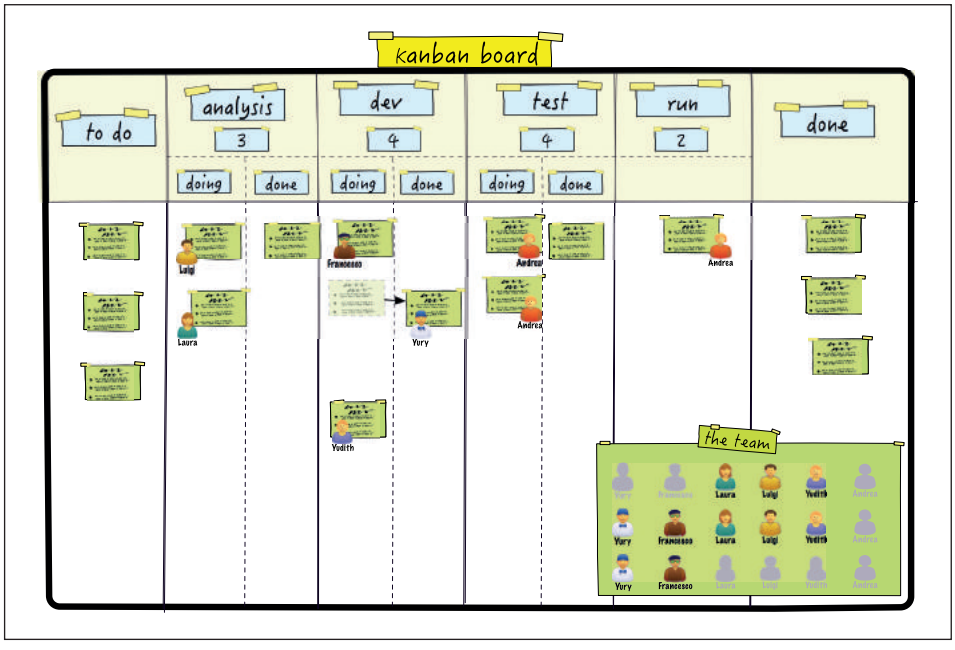


Figura IV.16. Yury, completato il suo lavoro, si rende libero per fare altro.

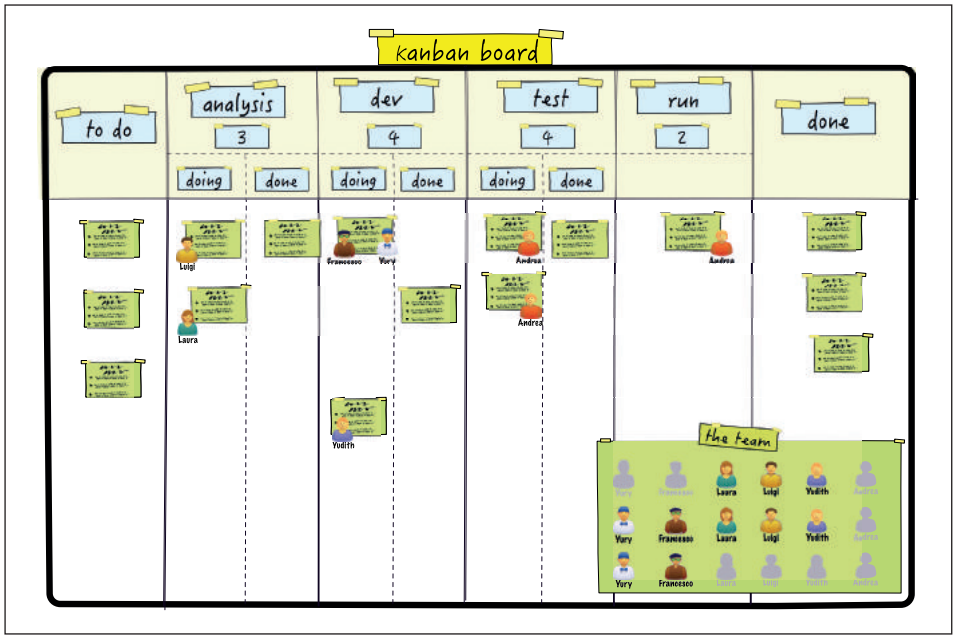


Figura IV.17. Yury, dopo aver terminato il suo task, non ne inizia uno nuovo, ma si mette in coppia con Francesco per aiutarlo a completare prima il suo lavoro.

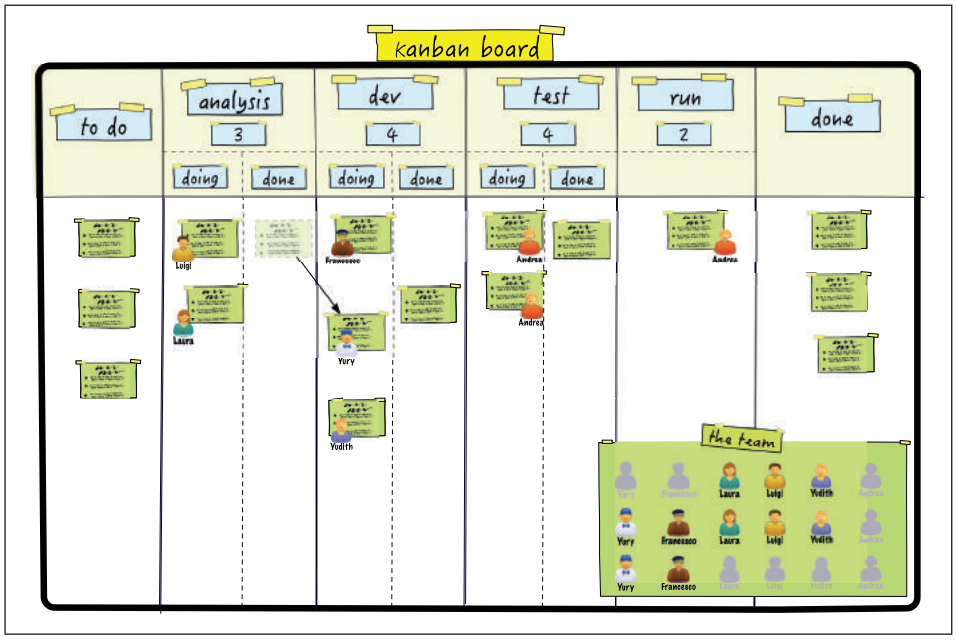


Figura IV.18. In alterntativa Yury invece potrebbe mettersi a lavorare un nuovo task.

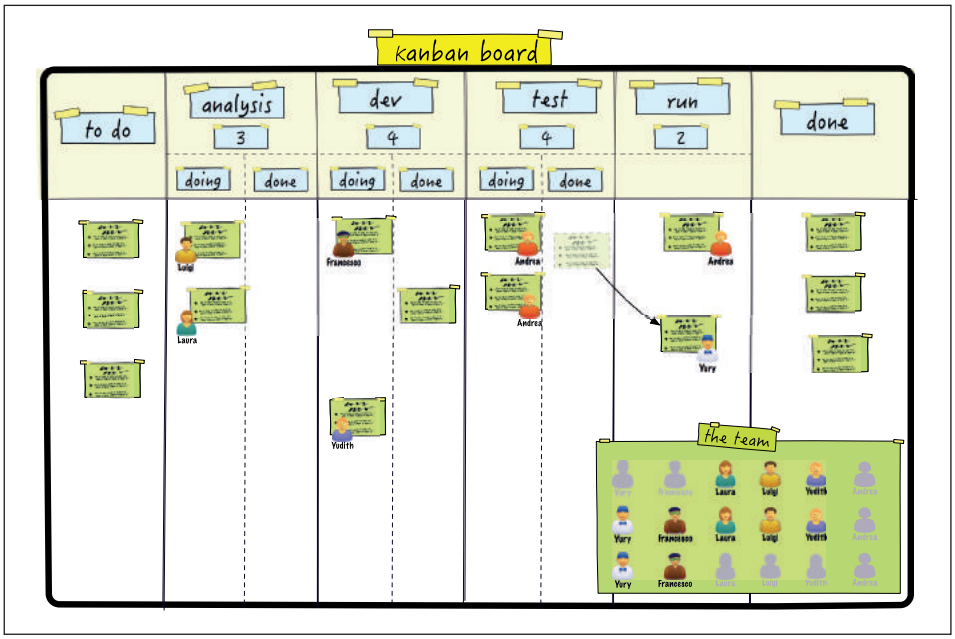


Figura IV.19. Se Yury è in grado di eseguire il test o il deploy, potrebbe mettersi a dare una mano ad Andrea per "scodare" le attività nella parte destra della board.

Dato che Yury è in grado di svolgere il lavoro dei suoi colleghi, a questo punto ha varie possibilità: per favorire il completamento delle attività, potrebbe mettersi a lavorare insieme a Francesco per completare più rapidamente il task di quest'ultimo (figura 17).

In alternativa, Yury potrebbe prendere in carico un nuovo task: per esempio (figura 18) potrebbe prendere un task dalla colonna **done** dell'analisi e metterlo in lavorazione per lo **sviluppo**.

Qualora Yury fosse in grado di svolgere più attività (oltre a sviluppare codice), potrebbe aiutare i colleghi per esempio nella fase di test o di deploy finale. Nella figura 19 collabora con Andrea nelle attività relative di test (parte destra della board).

Gestione delle emergenze

Normalmente la sequenza con cui le varie attività sono messe in lavorazione segue l'ordinamento dei cartellini nella colonna iniziale del **ToDo**: la priorità potrebbe essere stabilita in base all'importanza o all'urgenza della richiesta (come in un sistema di help desk) o semplicemente al tempo di arrivo.

A volte capita che il team debba interrompere la normale pianificazione per svolgere alcune attività ad alta priorità — spesso delle vere e proprie emergenze — come quelle relative a un bug bloccante in produzione o a una variazione da inserire urgentemente nel prodotto.

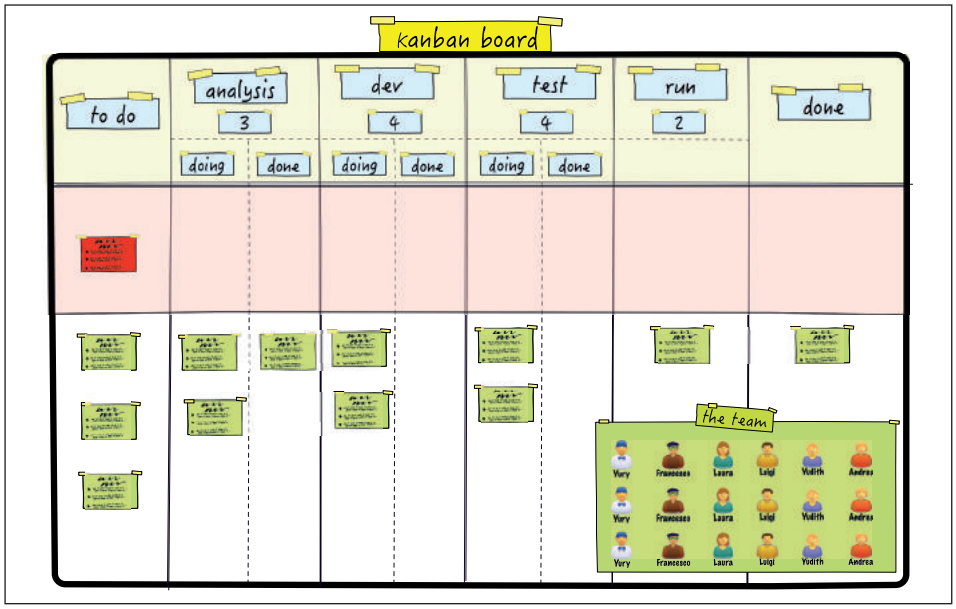


Figura IV.20. Una corsia di emergenza permette di gestire quelle attività che dovranno essere evase in tempi molto rapidi.

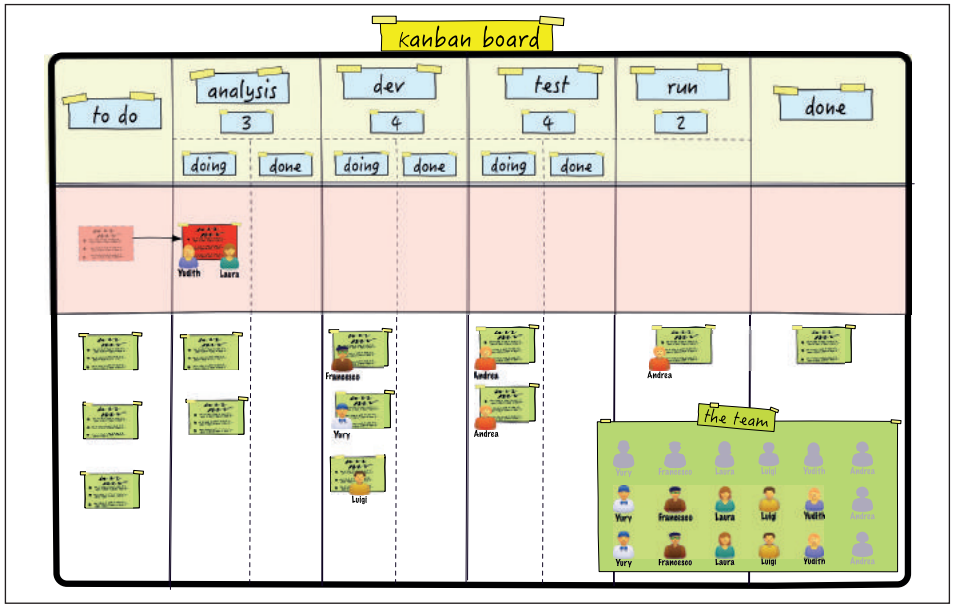


Figura IV.21. L'emergenza è presa in lavorazione. Due persone interrompono il loro lavoro "normale" e si dedicano a tempo pieno ad analizzare il problema. I rispettivi cartellini rimasti liberi sono lasciati nella loro posizione anche se di fatto occupano WIP senza che nessuno vi stia lavorando.

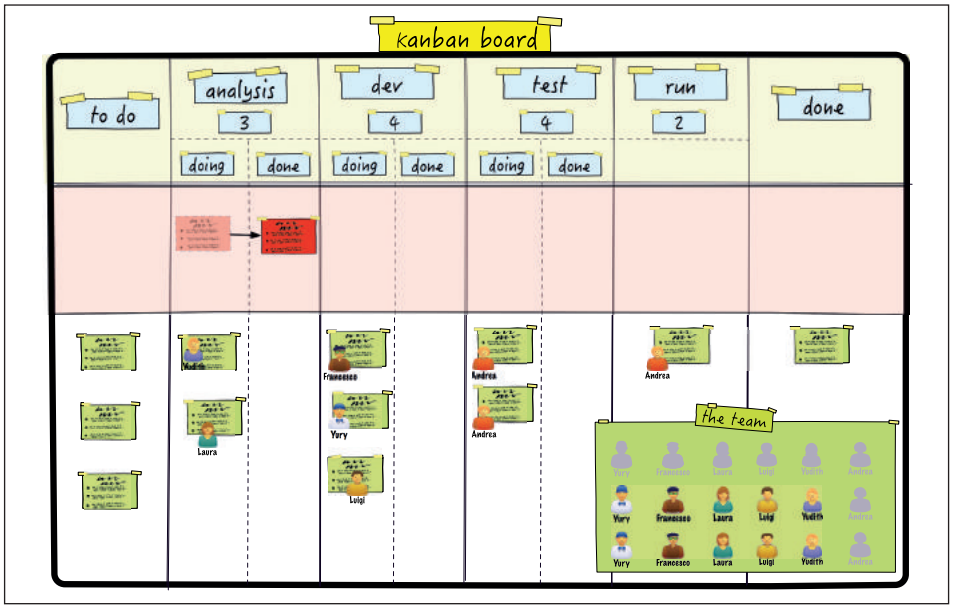


Figura IV.22. L'analisi è terminata; le due persone spostano il cartellino nella sottocolonna "done" e si rimettono a svolgere il proprio lavoro "normale" che avevano interrotto.

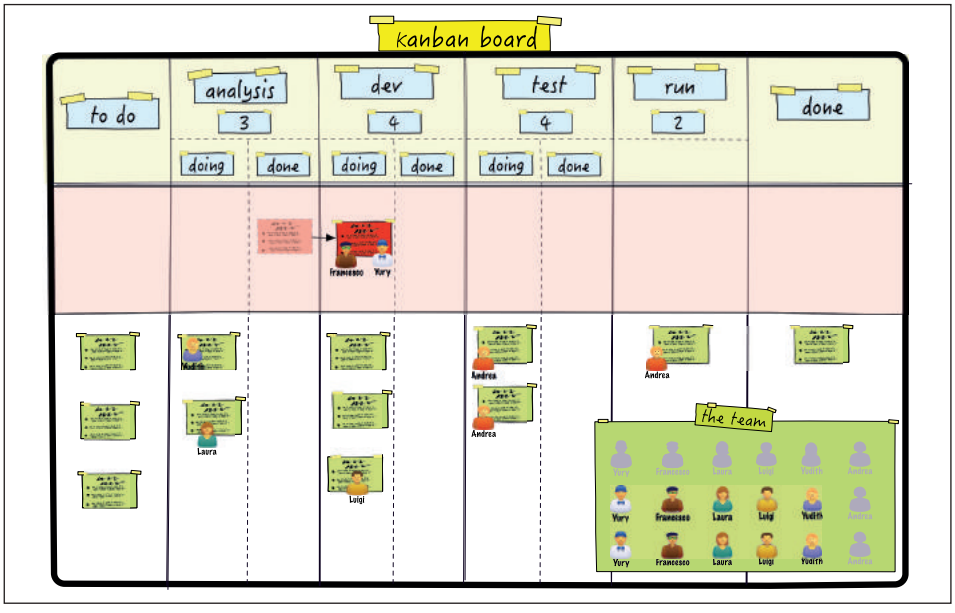


Figura IV.23. Adesso altri due membri del team interrompono il loro lavoro di sviluppo per implementare la funzionalità urgente. I passi successivi (verifica e rilascio) proseguono in modo analogo, fino a che il bug non è completamente risolto.

Per gestire queste situazioni, è utile apportare alcune piccole modifiche alla board; un modo è quello di inserire una apposita corsia orizzontale detta "corsia delle emergenze" o **Expedite Lane** (con riferimento alla corsia preferenziale riservata ai mezzi di soccorso) spesso identificata con un colore particolare (figura 20).

Analogamente anche i cartellini potrebbero essere colorati in base al livello di importanza, in analogia al codice colore che viene assegnato all'accettazione al pronto soccorso: rosso, giallo, verde, bianco o qualsiasi altra scala di colore ben comprensibile.

Nelle figure 21, 22 e 23 si può seguire la sequenza di azioni di una tipica procedura di emergenza.

Gestione dei difetti di lavorazione

Una variante del caso precedente si ha quando il team, durante la verifica di una attività in implementazione, trova un **difetto** prima che questo sia messo in produzione: si immagini il caso di un test che fallisce durante la verifica di una qualche parte del software (figura 24).

Un modo per gestire questa situazione potrebbe essere quello di **riposizionare** il cartellino corrispondente al test fallito, nella colonna della lavorazione necessaria per

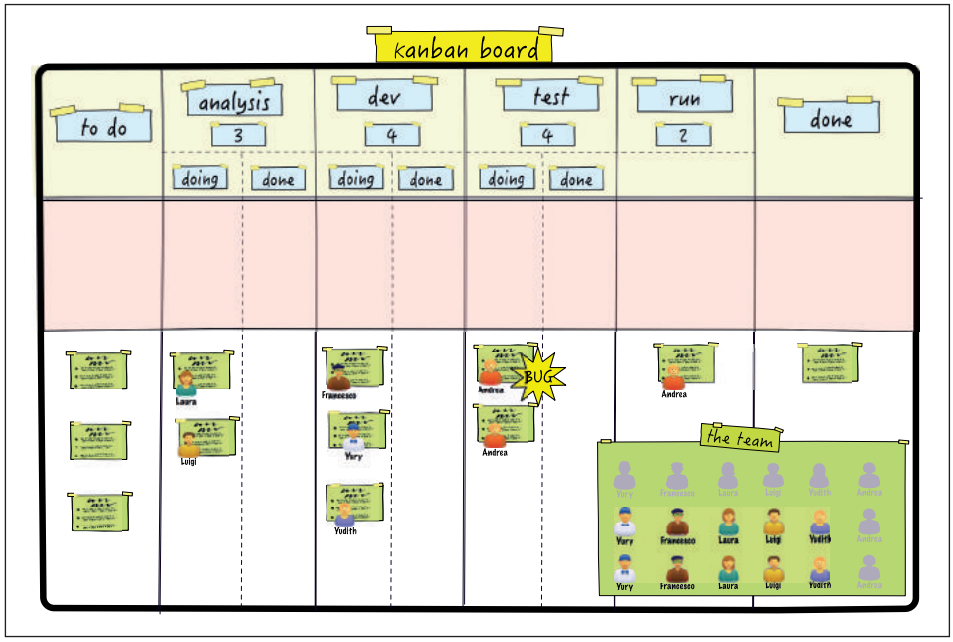


Figura IV.24. Caso simile al precedente ma con una sua specifica diversità è quello del difetto di lavorazione: il bug è trovato prima che la funzionalità sia messa in produzione. Una prima soluzione consiste semplicemente nel "retrocedere" il cartellino nella colonna di lavorazione necessaria per risolvere il problema.

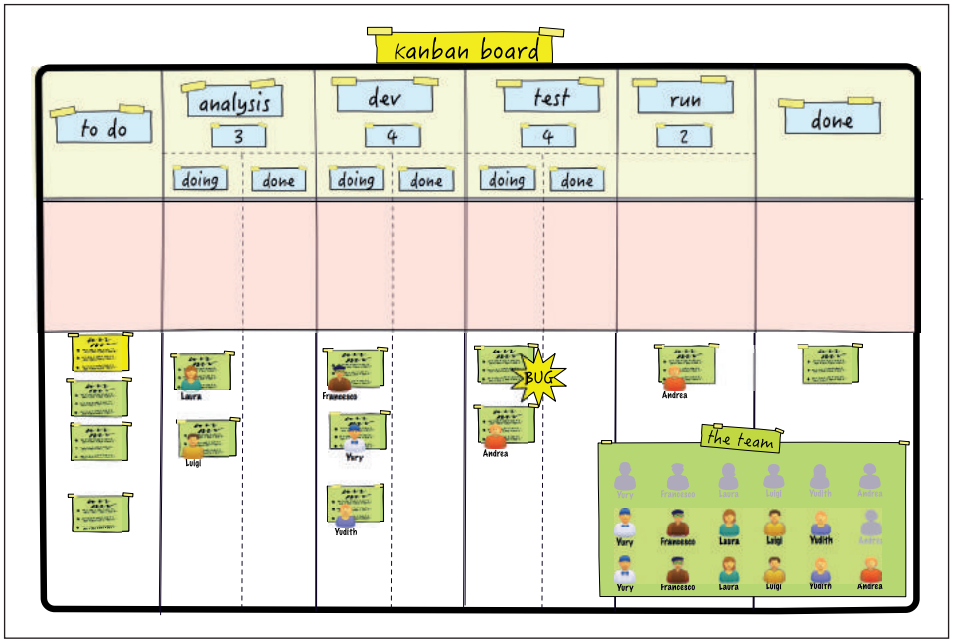


Figura IV.25. Un altro approccio alla gestione degli errori: in questo caso il cartellino associato all'attività che ha generato un errore viene "parcheggiato" (cartellino con segno rosso) nel punto dove si è verificato il problema, mentre uno nuovo "urgente" (in giallo), viene creato e messo in lavorazione. Il cartellino parcheggiato di fatto "consuma unWIP" per cui questa soluzione tende a portare a un ingolfamento del processo.

risolvere il problema (per esempio nuovamente in analisi o in sviluppo). Questa soluzione è quella più **semplice** da realizzare e anche la meno impattante a livello organizzativo: niente resta in sospeso nel punto dove ha generato l'errore e quindi si libera un WIP per altre attività che possono essere portate avanti.

Una variante, più scomoda da realizzare ma più **in linea** con i **principi agili**, potrebbe essere quella di lasciare il cartellino in errore nella sua posizione e di inserire in lavorazione un nuovo cartellino che andrà a "chiudere" il problema emerso (figura 25, 26 e 27).

In questo caso il cartellino con il bug rimane in attesa della soluzione del problema occupando un **WIP**, cosa che a tendere potrebbe portare al blocco del sistema: se ci sono molti cartellini "abbandonati" in attesa di una soluzione, niente altro potrà essere eseguito fino a che i problemi non sono stati risolti.

Questa strategia ha certamente un costo: secondo la filosofia Lean messa a punto da Ōno, questo costo deve essere da sprone non solo per risolvere il problema rapidamente ma, anche e soprattutto, per apportare le necessarie modifiche che evitino a quel problema di verificarsi nuovamente.

Interessante in tal senso la citazione di Teruyuki Minoura, ex presidente di Toyota Motor Manufacturing North America:

"Se si verifica un problema nello one-piece flow, si interrompe l'intera linea di produzione. In questo senso [la produzione Lean] è un sistema molto poco efficace. Ma quando si ferma la produzione, tutti sono costretti a risolvere immediatamente il problema: quindi devono riflettere, e riflettendo crescono e diventano membri migliori del team, e persone migliori."

Nella produzione di massa, invece, si fa largo uso di magazzino dei prodotti finiti e questo non fa emergere i problemi: è come se il sistema "nascondesse" i propri errori confinandoli nel magazzino invece di prenderli immediatamente in carico e per questo motivo un tale sistema di produzione è meno propenso al miglioramento continuo.

Suddivisione del flusso di lavoro: scatter & merge

Le board viste fin qui si adattano molto bene a quei casi in cui il processo è scomponibile in maniera **sequenziale**: ogni colonna corrisponde a una **fase** del processo di lavorazione; ogni attività, per essere completata, deve **passare** per **tutte** le **colonne** della board.

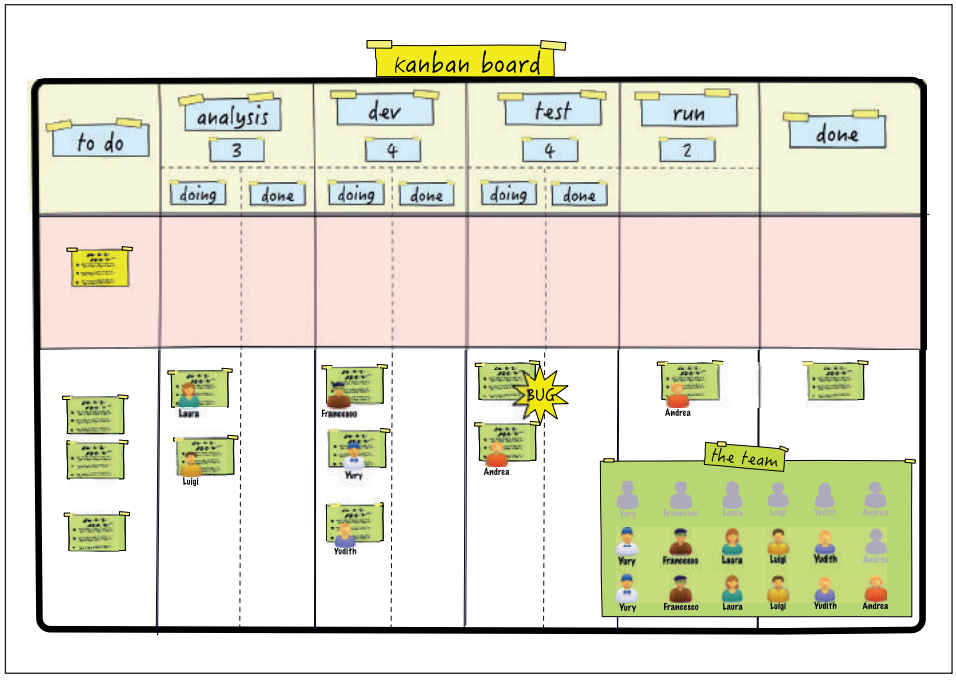


Figura IV.26. I passi successivi sono analoghi al caso precedente. Il cartellino giallo urgente viene messo in lavorazione con priorità nella corsia delle emergenze.

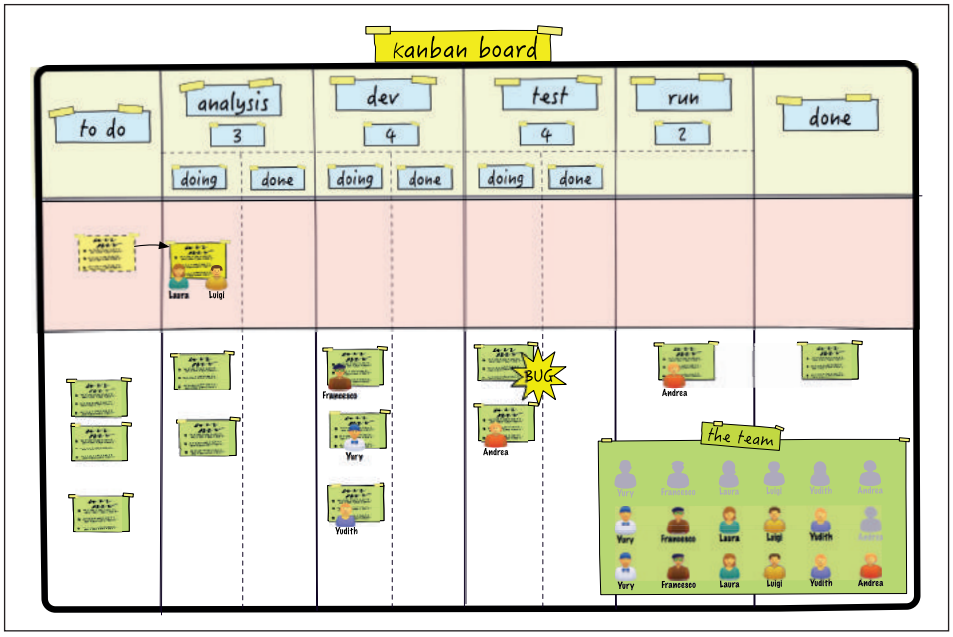


Figura IV.27. Anche in questo caso i membri del team potranno lavorare in coppia, oppure singolarmente.

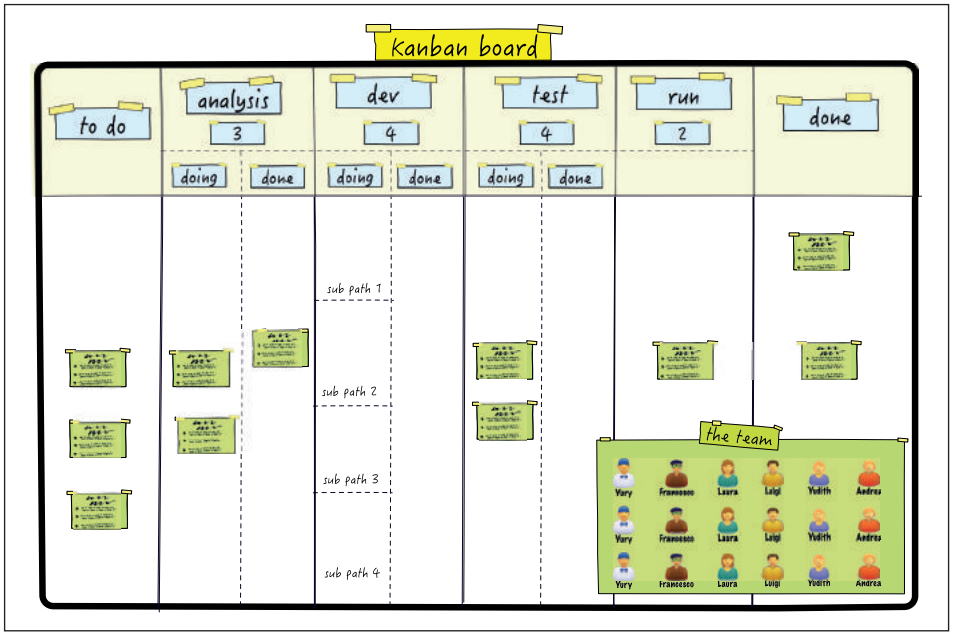


Figura IV.28. In questo caso lo sviluppo di un componente prevede differenti sottoattività.

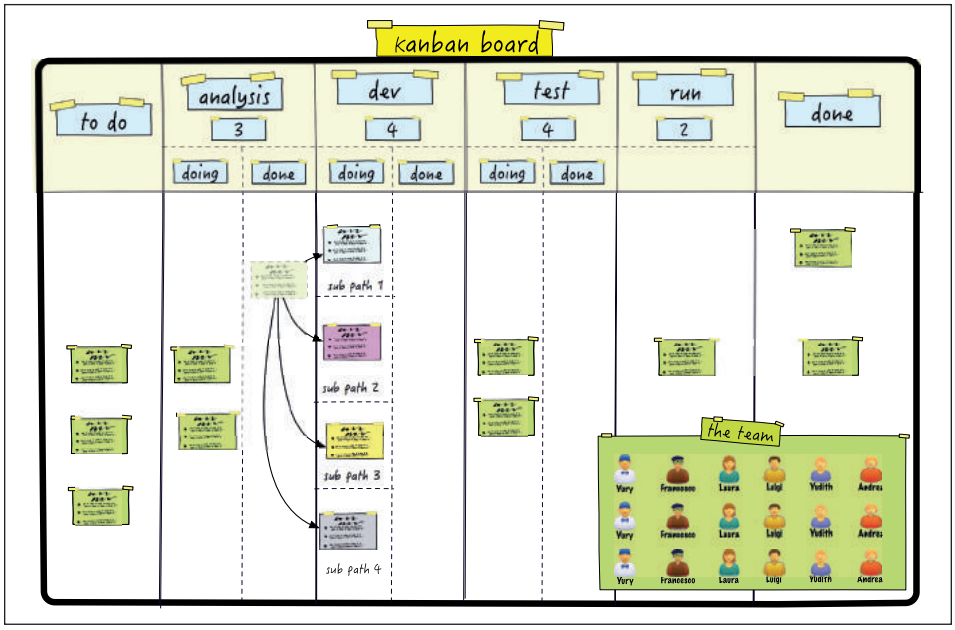


Figura IV.29. Ogni cartellino, per procedere nella lavorazione, viene "suddiviso" in altrettanti sotto-cartellini, uno per ogni sotto-attività.

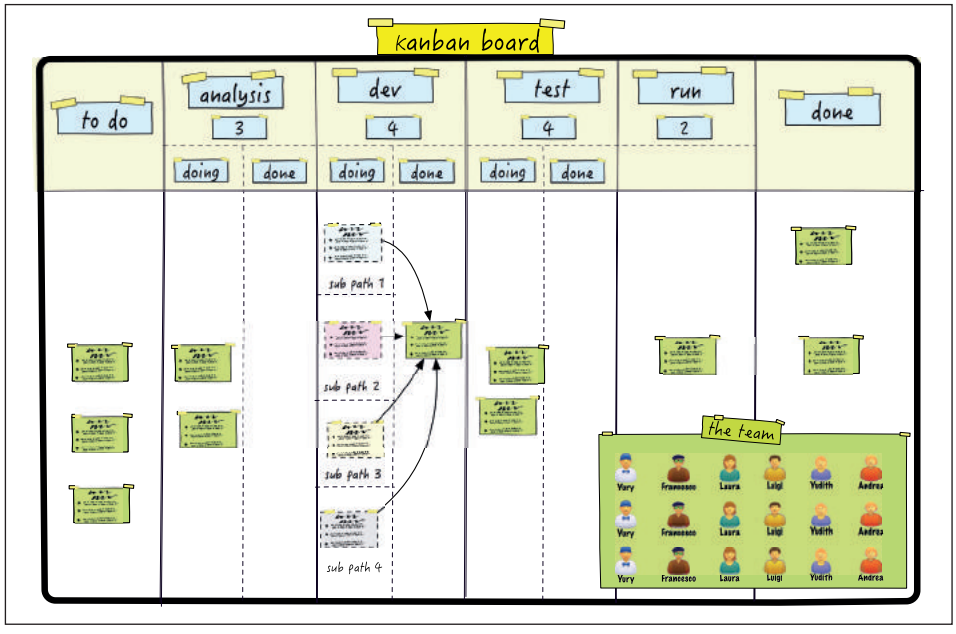


Figura IV.30. Solo quando tutte le sottoattività sono state eseguite, si potranno riunificare i diversi "sotto-cartellini" in un unico cartellino che continua il suo viaggio nel flusso della kanban board.

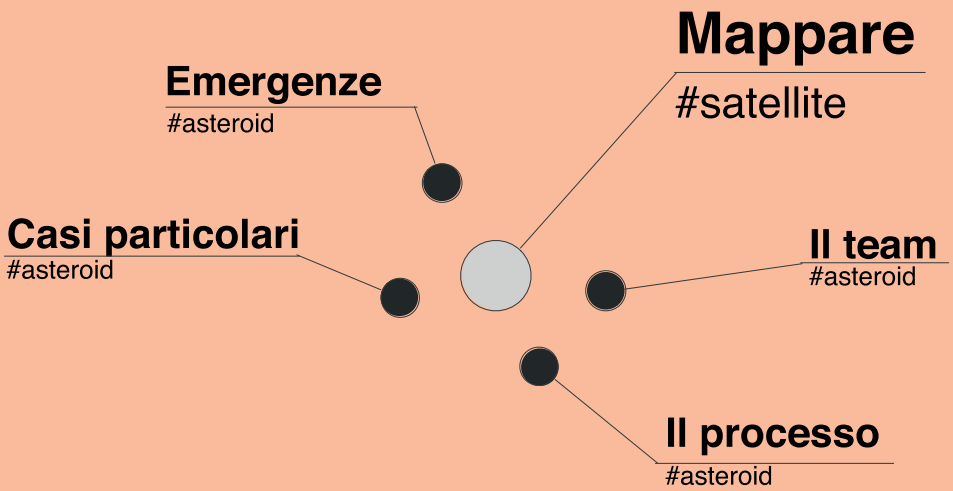
A volte però **non** è possibile **scomporre** il processo in **fette verticali**, perché alcune fasi della lavorazione possono richiedere attività che devono essere eseguite in parallelo. Si pensi per esempio alla produzione di un qualche componente che preveda lo sviluppo di una parte di **hardware**, di **firmware** di sistema e di **software** applicativo. Lo sviluppo potrebbe essere quindi diviso in queste tre linee di lavoro mentre il test potrebbe essere eseguito sul componente **dopo** che tutte le varie parti siano state **sviluppate** e **integrate**. Per poter gestire questo tipo di situazioni si può procedere a modificare la board introducendo delle corsie orizzontali solo nella colonna relativa allo sviluppo, come riportato nella figura 28.

In questo caso, un cartellino che arriva nella colonna di sviluppo, verrà **scomposto** in 4 **sotto-cartellini** che verranno poi **ricongiunti** in un **unico** cartellino man mano che lo sviluppo sarà terminato (figure 29 e 30).

Conclusioni

Abbiamo visto i passi necessari a creare una lavagna kanban. Nel capitolo successivo si vedrà come costruire una board che permetta di mappare un processo specifico.

Capitolo 3
Mappare il processo su una kanban board



Introduzione

Nei capitoli precedenti si è visto come è fatta una lavagna kanban e quali sono i principi Lean su cui tale strumento si basa. Siamo però partiti da un flusso ideale, creato a tavolino proprio per spiegare principi e meccanismi

Un passaggio che spesso preoccupa i neofiti è quello relativo alla creazione della **kanban board** per mappare un **processo reale** e permetterne una corretta ed efficace gestione. In realtà creare una board di questo tipo non è affatto complicato e anzi si possono seguire varie tecniche. Oltre a quello che raccontiamo in questo capitolo, esistono per fortuna molti libri e articoli sul tema, in grado di fornire validi suggerimenti.

In questo capitolo verrà mostrato, passo dopo passo, come costruire, una kanban board che rappresenti il processo in esame e che consenta di gestirne gli stati di avanzamento, di evidenziare i punti di miglioramento e altro ancora.

Passo numero 1: mappare il workflow

Mappare un **processo** di **produzione** tramite una **kanban** board è una attività che può essere svolta **iterativamente** in modo **incrementale**, aggiungendo ad ogni passaggio un dettaglio o una nuova parte.

Il suggerimento è di iniziare dall'individuazione del flusso del processo, o **workflow**; il flusso può essere definito anche in maniera grossolana, ma c'è un aspetto a cui va prestata la massima attenzione: cosa sta **dentro** e cosa sta **fuori** del processo, in modo di individuare approssimativamente il **contorno** del sistema che si vuole modellare.

In questa fase ci si può avvalere di strumenti grafici piuttosto semplici come un diagramma di flusso o un diagramma di stato UML (figura 31); utile in questa fase è l'utilizzo della tecnica del **Value Stream Mapping** per **identificare** i punti del flusso in cui si **genera valore**, punti che diverranno quelli su cui porre maggior attenzione nel processo di controllo tramite la kanban board.

In prima istanza non è importante definire i dettagli di ogni fase della lavorazione, ma ci si può limitare a disegnare il processo tramite una serie di **black box** in cui siano definite le **interfacce** di **input** e di **output** rispetto a sistemi esterni: lo scopo è separare ciò che partecipa alla realizzazione del prodotto finito, e che può quindi essere considerato come **parte integrante** del sistema, da ciò che invece può essere considerato un **elemento esterno**, con il quale il sistema deve **comunicare.**

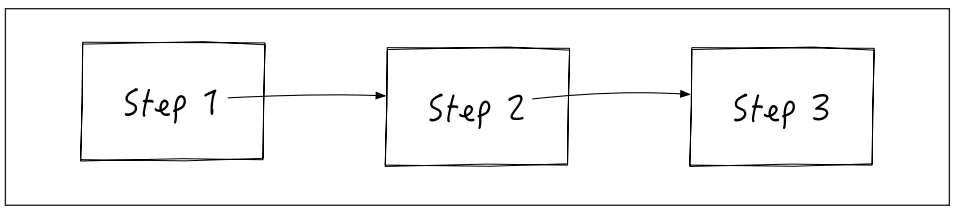


Figura IV.31 Si definisce un primo modello sintetico del processo.

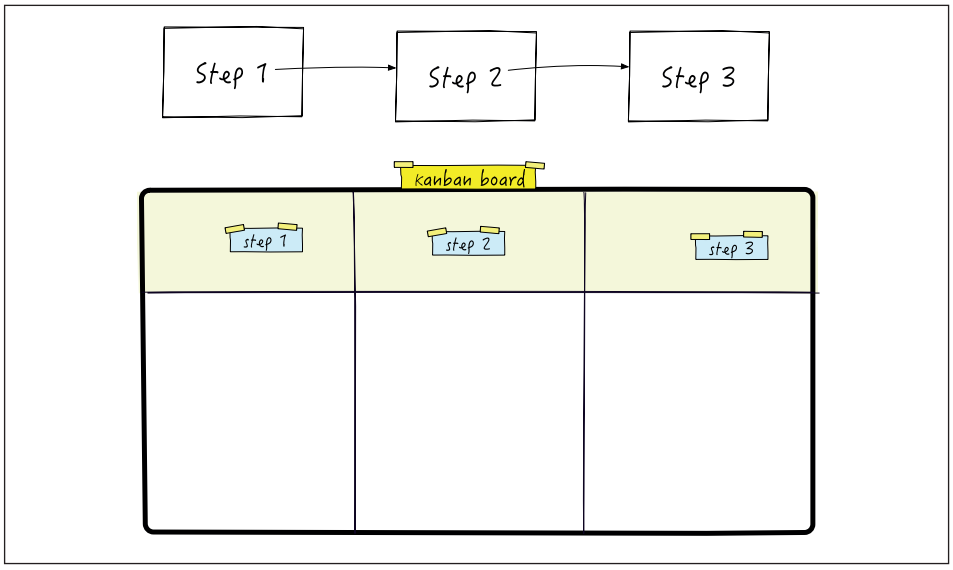


Figura IV.32. Dal workflow alla board.

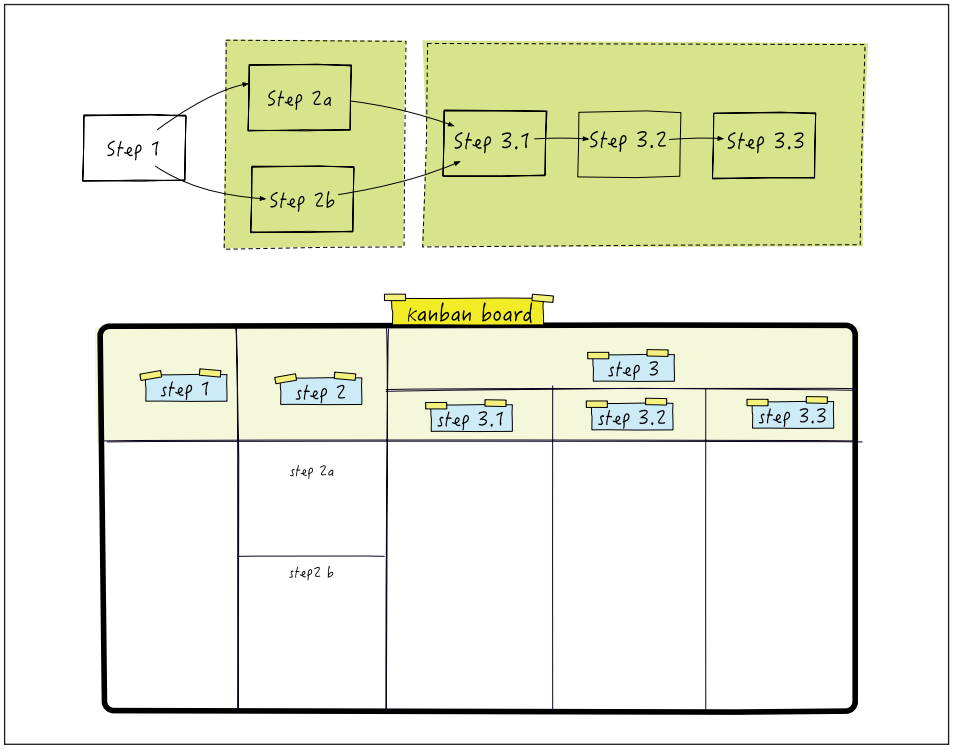


Figura IV.33. Cambia il livello di dettaglio del workflow, cambia la forma della board.

Disegnare la board

Dopo aver disegnato schematicamente il **workflow**, la realizzazione della kanban board, almeno in una sua prima rappresentazione approssimativa, è piuttosto semplice: ogni **passaggio** può essere rappresentato da una **colonna** della board.

Il livello di **dettaglio** della board dipende spesso dal grado di **precisione** che si vuole utilizzare per monitorare lo stato di avanzamento del processo di lavorazione: si potrebbe passare da un generico **ToDo**, **Doing**, **Done** — dove il **Doing** rappresenta tutto il processo di lavorazione nel suo complesso — a qualcosa di più dettagliato tramite l'inserimento di una **colonna** per ogni **fase** della lavorazione.

Non c'è una regola che possa aiutare a stabilire il livello di dettaglio nella definizione della board: quante colonne mettere, quali fasi rappresentare e quali invece accorpare con le altre. Una metrica potrebbe essere la frequenza o l'importanza degli eventi che si vogliono rappresentare con le colonne della board: mappare eventi rari o che contribuiscono marginalmente nella creazione di valore potrebbe introdurre complessità non necessaria. Utile in questo caso una analisi tramite il **Value Stream Mapping**.

Parlando in termini generali, la strategia è quella di aggiungere un dettaglio per volta e verificarne l'utilità.

Mappare la realtà del processo, non una sua rappresentazione ideale

Grazie alla sua capacità di **fornire visivamente** una **rappresentazione statica**, ma anche **dinamica**, dell'organizzazione, una kanban board è uno strumento estremamente efficace nel **mettere** in **luce pregi** e **difetti** del processo: non sempre tutte le persone

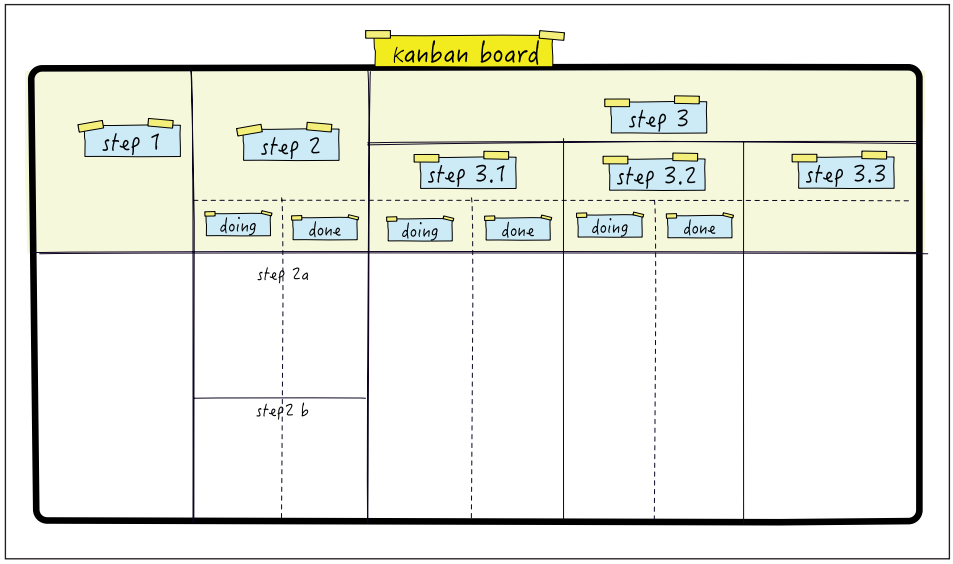


Figura IV.34. Si inseriscono le colonne buffer.

possono sentirsi a proprio agio con uno strumento che evidenzi in modo così diretto eventuali inefficienze nell'organizzazione.

Per questo motivo, affinché la diagnosi sia utile, conviene sempre rifarsi allo **schema reale** e non a quello "ideale" riportato magari in qualche documento ufficiale e utilizzato per una qualche presentazione in pubblico; capita alle volte che il processo messo **in atto** sia ben **differente** da quello **concordato** sulla **carta** in fase di progettazione. Ammettere questa discrepanza non è semplice specialmente se è una informazione non condivisa ufficialmente con i "piani alti".

Durante il processo di creazione della board è consigliabile rendere pubblico lo scopo del lavoro (trovare punti di inefficienza e migliorarli), per evitare che uno strumento di aiuto diventi causa di conflitti all'interno dell'organizzazione.

Passo numero 2: introduzione delle aree di buffer

Dopo aver disegnato le colonne, il passo successivo potrebbe essere quello di introdurre le aree di **parcheggio** o colonne **buffer** (vedi fig 34).

Passo numero 3: limitare la lavorazione in parallelo tramite il WIP limit

Dopo aver introdotto le colonne buffer, il passo successivo potrebbe essere dato dall'aggiunta, in testa alle colonne, dei **WIP limit**, in modo da vincolare il **numero** di **attività** che potranno essere svolte **in parallelo** all'**interno** di ogni **fase**.

Anche in questo caso non esistono regole o formule magiche per la scelta del limite del WIP: si consiglia quindi di seguire un approccio sperimentale per tentativi.

Il valore ottimale in genere si trova dopo alcune prove valutando, per ogni variazione, gli impatti sulle performance complessive sul sistema: per esempio si potrebbe individuare come metrica di riferimento il lead time medio necessario per completare la lavorazione delle attività prese in esame, prendendo il valore medio per avere una valutazione che tenga conto delle differenti tipologie di attività.

Uno strumento leggermente più raffinato e puntuale sulle varie fasi di lavorazione è quello che valuta il carico di lavoro di ogni step del processo: fasi troppo cariche (troppi di cartellini in lavorazione o in attesa di essere lavorati) o scariche del tutto (colonne vuote) sono indicatori di uno sbilanciamento sul processo, sbilanciamento che potrà essere "pareggiato" modificando i valori di limite del WIP. Ttipicamente un WIP troppo basso crea un ingorgo a monte e uno svuotamento a valle. Agendo sui vari **WIP limit** si può quindi rendere più armonico il bilanciamento del carico, cosa che in ultima analisi si ripercuote su un lead time minore.

Passo numero 4: definizione delle card

Oltre alla progettazione e al raffinamento della board, si può lavorare per aggiungere dettagli e informazioni ai **cartellini** corrispondenti ai task in lavorazione. Ogni cartellino dovrebbe contenere tutte le **informazioni significative** relative all'attività da

svolgere: dovrebbe essere presente un **titolo** che esprima in modo chiaro scopo e tipologia della attività, dovrebbe essere riportare l'**owner** o comunque il responsabile, la data di **nascita** (quando il cartellino è stato inserito nella lavorazione) e una **data** di **consegna** richiesta (oltre la quale il non completamento rappresenta un problema). Utile anche, qualcosa che ne specifichi la "classe" ossia l'importanza, urgenza o semplicemente il valore. Di seguito sono descritti nel dettaglio i campi utilizzati più frequentemente.

Titolo

Un buon **titolo** dovrebbe essere sintetico e dovrebbe essere sufficientemente chiaro e "parlante", affinché possa comunicare l'obiettivo dell'attività e il beneficio che porta al sistema o a una persona specifica.

Qualora si usi una board fisica, il titolo dovrebbe essere al massimo una o due parole scritte in grande in modo che siano visibili anche da lontano. Se necessario, ulteriori informazioni si possono inserire in un sottotitolo o in una descrizione.

Date di creazione e di completamento

Sono due campi molto importanti per capire l'**età** del cartellino, ossia quanto tempo si sta impiegando per completare la lavorazione. Volendo raffinare le indagini si potrà contare anche il tempo che un cartellino passa nelle colonne di buffer in modo da arrivare calcolare il **lead time** e il **cycle time**. La **data di completamento** ovviamente va compilata a termine della lavorazione.

Deadline

La data **ultima** di completamento ammissibile.

Created by / owner

Chi ha creato il cartellino o meglio chi ne è il "**proprietario**".

Priorità

La **priorità** si esprime con un numero o un codice che indica quanto sia urgente una determinata attività.

Segnalazione di impedimento (impediment warning)

A volte, attività importanti rimangono bloccate a causa di **impedimenti** di vario tipo. Se il blocco dovesse protrarsi oltre un certo limite, si può attaccare alla card un qualche elemento **distintivo** (per esempio un pallino adesivo rosso) in modo da **evidenziare** che c'è un problema che rallenta o blocca del tutto la lavorazione di un elemento critico.

Tipologia di attività

Il **Task Type** è riferito sia alla classe di servizio che a una specifica tipologia di attività all'interno del processo (p.e.: analisi o sviluppo o altro).

Descrizione

Una breve **spiegazione** del lavoro da svolgere, degli obiettivi, dei benefici e dei beneficiari.

Note

Annotazioni varie, formato libero.

Requisiti per definire l'attività come completata

In Kanban, i **requisiti di completamento** non sono nulla di diverso da ciò che in Scrum si chiamano **Acceptance Criteria**, vale a dire un elenco di criteri che permettano di stabilire quando un'attività si possa considerare davvero **completata**.

Devono essere "collegati" alle policy definite per stabilire come e quando un cartellino possa passare da una colonna alla successiva.

Storia

La **history** è una serie di appunti sugli eventi che hanno caratterizzato l'attività in questione, una sorta di cronistoria della "vita" della card. Dalla data della prima immissione nel processo, al completamento, alla revisione e successiva re-immissione in lavorazione per correzioni di vario tipo; utile anche segnare l'elenco delle persone che hanno contribuito alla lavorazione della card.

Passo numero 5: definizione delle classi di servizio

Una board realizzata seguendo i passi descritti fino a questo punto, potrebbe essere uno strumento già molto utile per esempio per controllare lo stato di avanzamento delle attività di un team di sviluppo. Un ulteriore e importantissimo passaggio è quello di introdurre la possibilità di gestire le **diverse tipologie** di attività.

Nella realtà, infatti, alcune sono molto **urgenti** tanto che ogni minuto speso per la soluzione costa all'azienda molti soldi: si pensi a un bug bloccante in produzione.

Altre invece possono attendere per essere messe in lavorazione quando non ci sono urgenze da completare.

Infine ci sono attività a bassa priorità che possono essere lavorate solo quando non ce ne sono altre in coda.

La classificazione delle attività potrebbe essere fatta anche in funzione di altri parametri, per esempio considerando la persona che può svolgere quel lavoro.

In Kanban si parla di **classi di servizio**, per indicare la differente modalità di presa in carico in funzione dell'urgenza o dell'importanza.

Identificare le classi di servizio

Per inserire la gestione delle classi di servizio all'interno di una kanban board, la prima cosa da fare è identificare le varie tipologie di attività: per esempio in un progetto software si potrebbero considerare le seguenti:

- implementazione di nuove funzionalità;
- bug fixing a bassa priorità;
- bug fixing ad alta priorità;
- documentazione;
- attività di formazione a supporto degli utenti;
- setup ambienti e installazione software di supporto;
- studio per realizzazione di prototipi.

Per ognuna di queste categorie, si prova a definire le priorità, per esempio valutando il **Cost of Delay**, eventuali **SLA** (**Service Level Agreement**) contrattualizzati o altro ancora.

Fatto questo si possono creare delle **macrocategorie** (dette appunto **classi di servizio**) dentro le quali far ricadere le varie attività. Ogni classe dovrebbe avere una **priorità** associata oltre alla **policy** di gestione.

Per esempio, si potrebbero definire queste tre classi:

- **Classe 1** –priorità **bassa**. Costo del delay non influente. Trattamento specifico consiste nel mettere in fondo al backlog, lavorare se non c'è altro da fare. Quando la coda delle attività di tale classe supera le 5 unità, mettere in lavorazione quella più anziana.
- **Classe 2** – priorità **standard**. Costo del delay varia in base agli accordi contrattuali (rilasci ad ogni iterazione), è significativo ma non enorme. Trattamento specifico: lavorazione urgente ma non bloccante.
- **Classe 3** – priorità **alta**. Costo del delay: ogni giorno passato in lavorazione costa 1000 €. Trattamento specifico: lavorazione urgente e bloccante; ogni altra attività deve essere interrotta. Se necessario, più persone possono mettersi a lavorare su un task di questo tipo.

Classi di servizio per armonizzare le attività

Utilizzare le **classi di servizio** permette di applicare una gestione **coerente** delle varie attività, per esempio razionalizzando la modalità di intervento in base alle policy specificate per una determinata classe. Da notare che l'introduzione di politiche di gestione particolari, per limitare il **cost of delay**, porta contemporaneamente altri costi: bloccare la produzione o interrompere il lavoro di più persone per risolvere un bug in produzione ha ovviamente un costo, anche se nella maggior parte dei casi non è evidente e quindi si tende a **sottovalutarlo**. Grazie all'osservazione delle **metriche** che si possono inserire in una kanban board, è comunque possibile fare delle valutazioni in tal senso.

Dal punto di vista operativo, le **classi di servizio** possono essere introdotte tramite un **codice colore** particolare per i cartellini da apporre alla kanban board, oppure tramite la creazione di corsie (**swimlanes**, figura 35) specifiche per ogni classe.

Passo numero 6: migliorare la board tramite le metriche

L'inserimento delle **colonne buffer**, la definizione dei **WIP limit**, così come la definizione delle varie **colonne** della board, sono decisioni che dovrebbero essere sempre

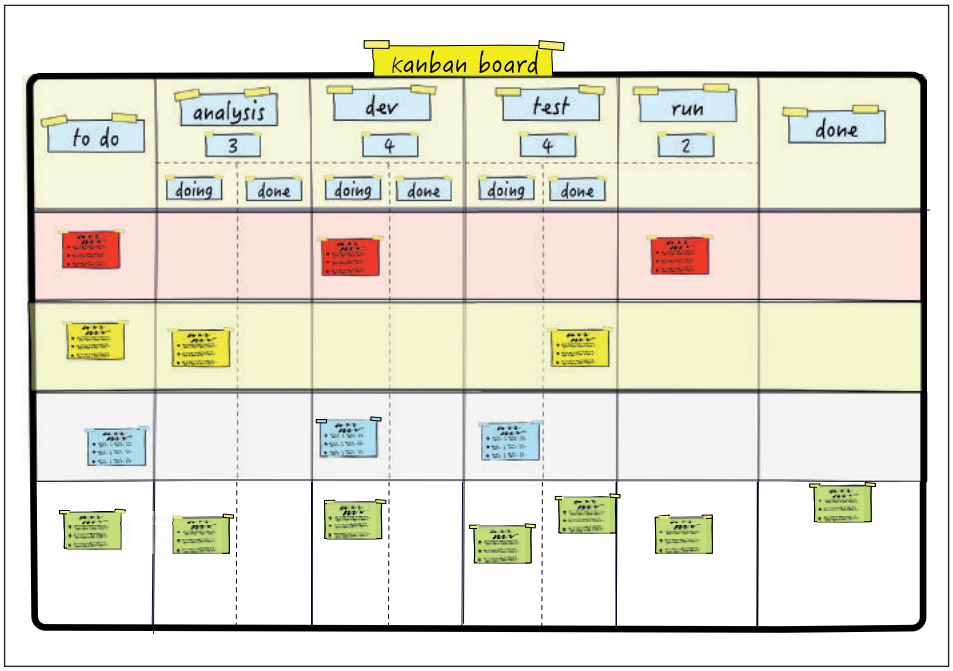


Figura IV.35. Sulla kanban board, è possibile segnalare l'appartenenza a una determinata classe di servizio grazie a specifici codici colore (rosso, giallo, verde, etc.) applicati ai cartellini, o per mezzo di "corsie dedicate" a ciascuna classe di servizio.

soggette a valutazioni e riconsiderazioni sulla base dell'osservazione diretta del sistema e dell'efficacia della board stessa.

Spesso la configurazione migliore si verifica grazie alla naturale capacità che ha una board di fornire una **rappresentazione visuale** del sistema. Altre volte invece si rendono necessari approfondimenti basati su sperimentazioni e analisi retrospettive, successi e fallimenti. Non esiste una soluzione buona a prescindere, ma si deve trovare di volta in volta la configurazione migliore. Misurare quindi la risposta del sistema è un modo oggettivo per valutare l'efficacia delle varie prove.

Oltre alle metriche di cui si è già parlato, **lead time** e il **cycle time, molto utilizzate anche** il **Cumulative Flow Diagrams** (**CFD**), il **Defect Rate** e il **Blocked Items**. La scelta della metrica da usare è spesso funzione del tipo di indagine che si vuole fare ma soprattutto del grado di maturità dell'organizzazione.

Tipicamente si consiglia di inserire l'uso delle metriche in modo graduale, a mano a mano che l'organizzazione acquisisce capacità sia di lettura che — soprattutto — di intervento.

Avremo modo di parlare di **metriche** nel prossimo capitolo, quando affronteremo nello specifico questo argomento.

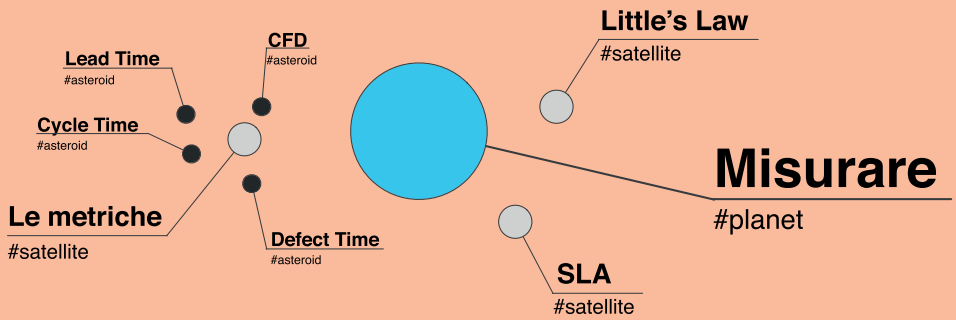
Conclusioni

In questo capitolo si è affrontato il tema della **mappatura** di un **processo** tramite la definizione di una board. Gli step visti sono i seguenti.

- mappare il workflow;
- introdurre le aree di buffer;
- limitare la lavorazione in parallelo tramite il WIP limit;
- definire le card;
- definire le classi di servizio;
- migliorare la board tramite le metriche.

È importante ribadire una volta di più che tutte le scelte che si fanno in questa fase della progettazione dovranno sempre essere **valutate** in funzione della capacità da parte della board di rispondere alle necessità di **monitoraggio** e di **miglioramento** del **processo**. Non esiste **una** soluzione ottimale, ma ogni decisione dovrà poi essere valutata e misurata sul campo sulla base dei risultati ottenuti. L'introduzione delle **metriche** può essere in tal caso un ottimo modo per poter valutare l'efficacia dello strumento adottato. Ne parleremo nel prossimo capitolo.

Capitolo 4
Misurare le performance



Introduzione

L'uso delle metriche è certamente molto importante sia per misurare le performance del team, sia e soprattutto per valutare l'efficacia di una modifica alla board.

In entrambi i casi, **misurare** permette di comprendere la bontà delle azioni intraprese: differenti strategie di lavoro, differenti atteggiamenti delle persone, differenti dettagli introdotti nella board.

Misurare con efficacia non è facile: occorre individuare una o più **metriche adatte** e misurare con rigore se si vogliono ricavare **informazioni utili** e non solo cifre. Se si vuole migliorare, non è tanto il valore puntuale di un KPI (*Key Performance Indicator*) ad essere significativo, ma l'andamento su medio-lungo periodo, ricavabile tramite **serie storiche** affidabili.

I KPI sono certamente utili per delineare un piano evolutivo, ma occorre **evitare** di **eleggere i numeri** a ruolo di **guida indiscussa** in una transizione agile. L'**ambiente** e il **contesto**, le **persone**, il **prodotto** sono tutti aspetti che spesso non è possibile misurare ma che rappresentano il **baricentro** di un'organizzazione e della sua evoluzione.

Misurazioni e obiettivi

Ogni organizzazione ha un suo livello di performance realistico (ottimale?): **spingere oltre** tale limite a volte non solo non porta alcun beneficio, ma anzi può essere fonte di una **degradazione** delle prestazioni o avere **conseguenze** sia sulle **persone** (stress, peggioramento delle dinamiche di gruppo, abbassamento della qualità della comunicazione) che sul **processo** (aumento dei difetti, peggioramento della qualità, ritardi nella consegna e altro ancora).

Per questo motivo è importante valutare l'**andamento** della serie storica, per enfatizzare i miglioramenti, ma anche per individuare prontamente un peggioramento.

Tenere stabili le metriche è fondamentale per avere valori comparabili; è altresì vero che se non si nota alcuna variazione del KPI scelto, è probabile che si stia guardando dalla parte sbagliata.

Prendendo il grafico della serie storica di un qualsiasi KPI, la discesa da un picco di massimo rendimento dovrebbe mettere in guardia: perché le cose stanno peggiorando? Cosa abbiamo fatto di diverso? E se volessimo invertire la tendenza cosa potremmo fare di diverso? Quali altre opzioni possiamo provare?

Metriche come strumento di allineamento

Le metriche sono quindi lo **strumento** per misurare gli effetti di una sperimentazione. Introdurre un **KPI** nel processo comporta quasi sempre un **costo diretto** — il dover effettuare le misurazioni — ma anche **indiretto**, legato per esempio alle conseguenze organizzative e politiche che si possono evincere dai risultati.

Chiedere alle persone di misurare qualcosa legato al loro lavoro può essere a volte una richiesta scomoda, un jolly da giocare con attenzione. La scelta di una metrica dovrebbe quindi tenere in considerazione anche questi aspetti: è consigliabile sceglierne

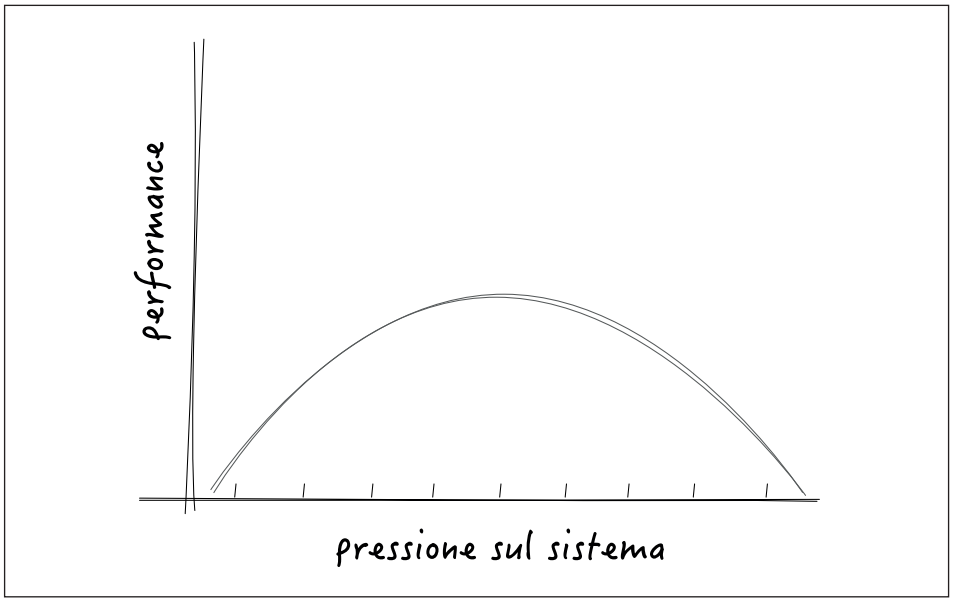


Figura IV.36. Provando a stressare un qualche fattore operativo nel team, le performance potrebbero, sul breve periodo, migliorare; insistendo oltre un certo limite, invece degradano bruscamente, con un trend molto più ripido di quello riportato in figura. Attenzione che le azioni che si devono compiere una volta superato il limite devono essere diverse per tipo e intensità: ossia è molto difficile recuperare un contesto che ha superato la soglia massima.

una compatibile con l'organizzazione, semplice da utilizzare e applicabile con costanza in maniera continuativa.

Le metriche dovrebbero essere quindi **strumenti** di **allineamento**, più che un criterio di valutazione del successo: allineamento in modo che **tutti** possano essere **informati** dello stato delle cose.

Quali metriche scegliere?

Dopo questa doverosa premessa, resta da rispondere alla domanda principale: cosa si misura? Fra le varie opzioni possibili, un buon punto di partenza, potrebbe essere quello di scegliere una metrica fra quelle che sono utilizzate nella maggior parte dei casi: **Cumulative Flow Diagram**, **Cycle Time/Lead Time**, **Defect Rate** e **Blocked Items**.

Cumulative Flow Diagram (CFD)

Il **Cumulative Flow Diagram** (CFD) è un diagramma con il quale si rappresenta lo stato di **avanzamento** della lavorazione: in un certo senso è paragonabile al **Burn Down Chart** rispetto al quale viene spesso preferito a causa della capacità di fornire una visione aggregata di varie metriche e con un dettaglio maggiore. Per questi motivi,

se da un lato il Burn Down Chart rappresenta lo strumento spesso utilizzato dai team alle prime armi, il **CFD** è invece adottato all'interno delle organizzazioni o dei team agili più maturi.

Il CFD è un diagramma in cui sull'asse orizzontale è riportato il **tempo** espresso tipicamente in **iterazioni** (se si adotta Scrum) o semplicemente **giorni** (opzione tipicamente utilizzata se si lavora in Kanban). In verticale invece è riportato il numero di cose da fare.

Come si può notare nella figura 37, il CFD mette a confronto nello stesso quadrante cartesiano la **curva del backlog**, ossia la totalità delle cose da fare, con i diagrammi corrispondenti alle varie **sottoattività** necessarie per completarle. L'ultima linea rappresenta l'ultima **attività** necessaria al completamento, in questo caso il **deploy finale**. Quando la prima curva **coincide** con l'ultima significa che tutte le cose da fare che erano in lista sono state completate.

L'**inclinazione** di queste linee è utile per comprendere come si comporta l'organizzazione o semplicemente come procede il progetto: per il backlog rappresenta la velocità con cui nuove cose sono messe nella lista delle cose da fare, mentre le altre curve indicano la performance puntuale delle varie sotto attività. Se l'inclinazione della curva del backlog è costantemente maggiore di quella del finito, significa che si stanno aggiungendo più cose da fare di quante il gruppo ne riesca realisticamente a realizzare.

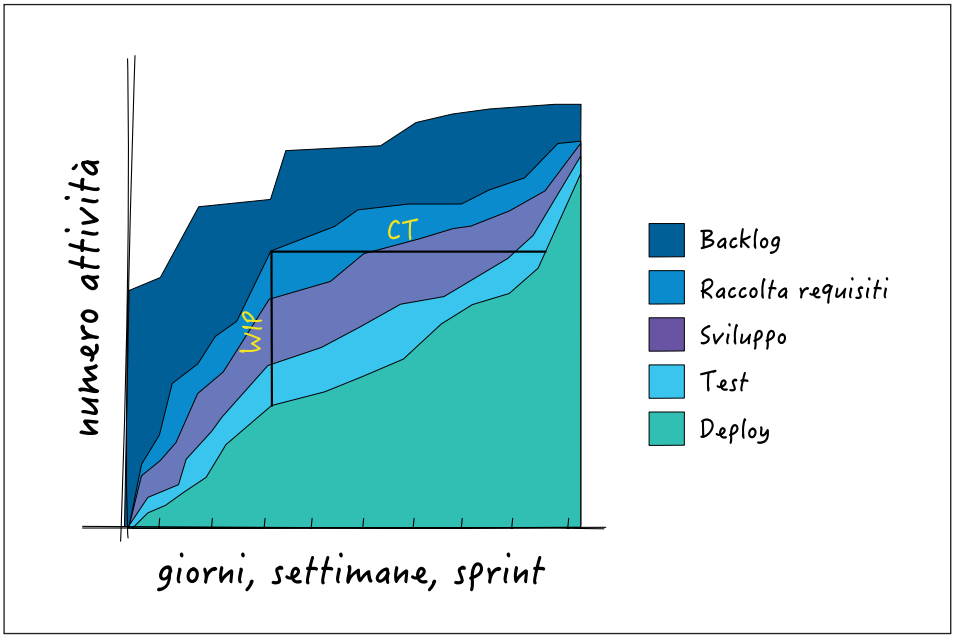


Figura IV.37. Il Cumulative Flow Diagram offre una visione d'insieme dell'andamento del totale delle cose da fare e delle attività necessarie per completare tali attività.

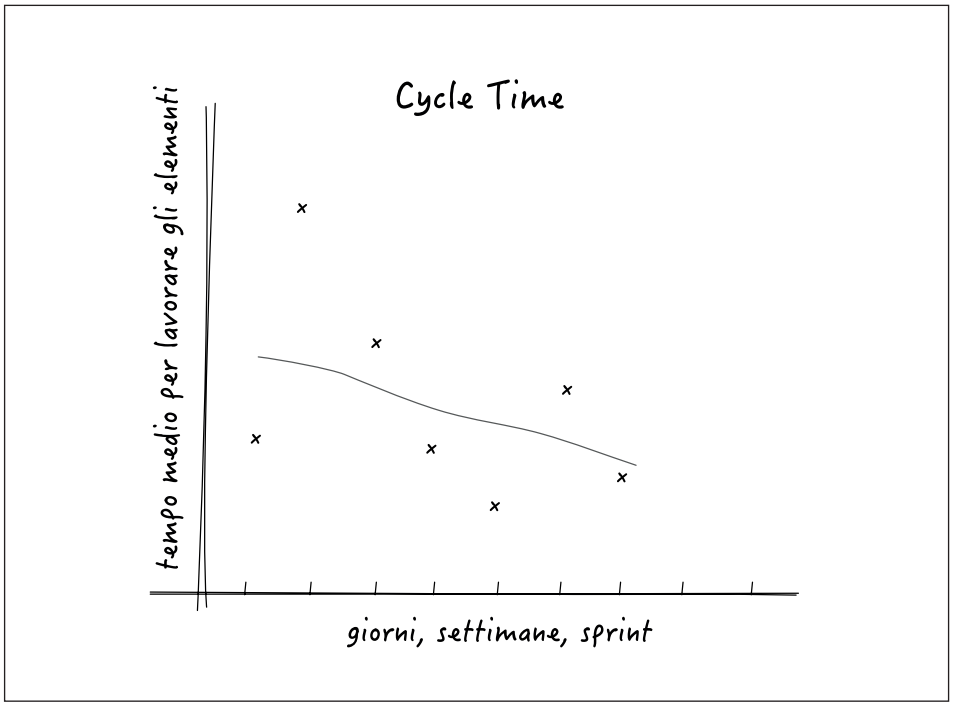


Figura IV.38. Riportando i punti dei vari Cycle Time su un piano cartesiano, si può visualizzare sia il trend che il grado di "sparpagliamento" ossia, usando un termine statistico, la varianza.

La **proiezione verticale** (nel disegno indicata con **vettore del WIP**) fra la curva relativa alla prima fase della lavorazione (in questo caso la raccolta delle specifiche, curva grigia) e quella corrispondente all'ultima fase di lavorazione (in questo caso il deploy finale, curva celeste,), rappresenta il numero di attività in lavorazione in ogni momento. Se il vettore del WIP aumenta, è probabile che sia in atto un rallentamento o, peggio ancora, un **collo di bottiglia**.

Nella figura per esempio. l'andamento del vettore del WIP è "sano": dopo un aumento iniziale (fenomeno naturale, visto che spesso è necessario il completamento di molte attività tecniche per poter rilasciare in produzione), diminuisce fino quasi ad azzerarsi.

La proiezione orizzontale rappresenta invece il tempo che impiega il team per completare la lavorazione di un elemento, ossia il cosiddetto **Cycle Time** medio (nel diagramma: linea **CT**). Se invece si traccia una linea che parte dal **tempo zero** si può calcolare il tempo complessivo che un elemento "trascorre" nel backlog prima che la sua lavorazione sia completata, ossia si ottiene il **Lead Time** medio (**LT**): da questo grafico si comprende meglio il concetto di questa grandezza, che è data da "tempo di attesa" più "tempo di lavorazione".

Un diagramma di questo tipo offre numerose informazioni circa le performance del gruppo di lavoro: il **CT** per esempio fornisce informazioni interessanti se si vuole capire come funziona il **team**; il **LT** invece, dato che rappresenta il tempo per evadere completamente una richiesta, è utile per valutare il livello di **servizio** offerto dall'organizzazione al cliente finale.

Per compilare un CFD è sufficiente che il gruppo di lavoro tenga traccia delle tempistiche di lavorazione (normalmente inizio e fine di un'attività) delle varie attività e sottoattività.

Andamento del Cycle Time e della varianza

Il CFD consente di ricavare il valore puntuale del Lead Time e il **Cycle Time** in un determinato momento. In ottica di miglioramento continuo è utile prendere in esame sia i **valori** misurati che il **trend** di tali metriche (figura 38).

Imparare a "confezionare" attività comparabili dal punto di vista dell'**effort** è un obiettivo comune a molte organizzazioni, dato che aiuta a stabilizzare il flusso e semplificare le attività di previsione: il trend della varianza è quindi un ulteriore importante indicatore per valutare l'efficacia degli sforzi del team nello stabilizzare il processo.

Da notare che il **Cycle Time** è un indicatore *a posteriori* e per questo dovrebbe essere utilizzato **in combinazione** con il **CFD** per avere una visione più completa.

Numero di elementi bloccati

Dato che il **tempo** di **esecuzione** di una attività (**CT** o **LT**) è proporzionale alle performance complessive, è evidente che ogni attività **bloccata** o **bloccante** ha ripercussioni negative sulle prestazioni dell'intero ciclo produttivo. Il **numero** di elementi **bloccati** è spesso ricavabile da altri strumenti come il **CFD** o il **CT**, anche se spesso i team preferiscono evidenziarne numero e tempo di risoluzione in modo esplicito, tramite metriche e diagrammi appositi.

Spesso il team utilizza **icone** dai **colori vivaci** e **rappresentativi** — rosso per i blocchi, arancione per gli impedimenti potenzialmente bloccanti — (figura 39) da attaccare sui cartellini della board: l'osservazione della board e della "macchia colore" permette di capire con un colpo d'occhio se ci sono problemi evidenti.

La figura 40 mostra un grafico dove sono riportati sia il numero di elementi bloccati che la media per la risoluzione del problema.

Defect Rate

Se il **numero** dei **difetti prodotti** da un team fornisce una indicazione puntuale e in tempo reale della qualità (bontà del processo, capacità delle persone, affidabilità dei test), monitorare l'andamento dei difetti prodotti è utile per avere una qualche indicazione sui benefici derivanti dalle eventuali modifiche apportate all'interno dell'organizzazione.

Come per le altre metriche viste fino ad ora, non è il valore puntuale a essere interessante quando il **trend**: se aumenta, perché aumenta? È perché è stato cambiato qualcosa

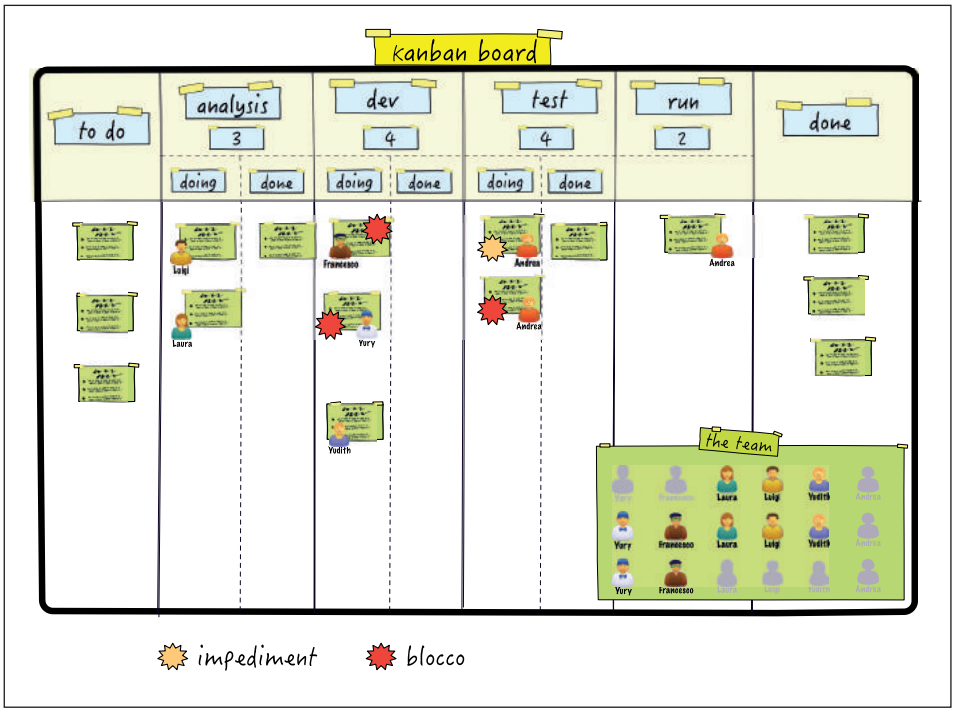


Figura IV.39. Apporre dei simboli colorati alle attività bloccate o con impedimenti è un modo semplice ed efficace per evidenziare se c'è un problema da qualche parte.

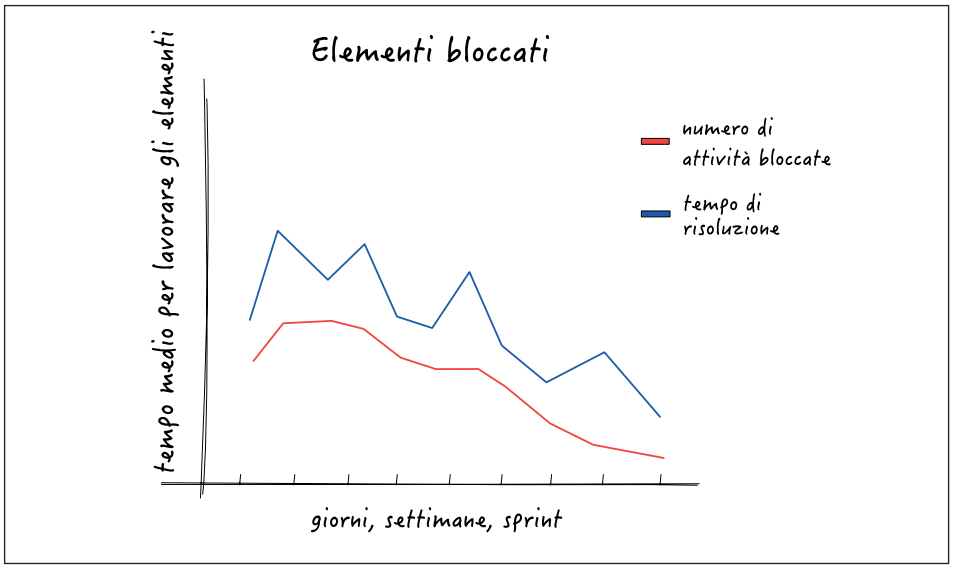


Figura IV.40. Grafico degli elementi bloccati e del tempo medio di risoluzione.

nel processo di lavorazione, oppure perché sta aumentando la complessità del progetto? Di che tipo sono gli errori che si presentano? Sono difetti puntuali o di interdipendenza fra le varie parti? Sta peggiorando la struttura complessiva? L'architettura non è adatta al tipo di progetto che si sta realizzando? Manca un'adeguata politica di governance delle parti rilasciate? È stato deciso — e comunicato a tutti — cosa fare in caso di errore?

Anche quando le cose migliorano è utile interrogarsi sul perché: ci sono segni evidenti che il miglioramento sia dovuto a qualche azione o a elementi specifici, oppure è un caso? Dobbiamo aspettarci che possa peggiorare? Rispondere a queste domande può essere un ottimo punto di partenza per intraprendere le azioni opportune.

Sempre in tema di valutazione del trend, un paio di interessanti varianti sono il **numero dei difetti aperti** e il **tempo di risoluzione medio**: se un incidente può accadere per mille motivi, la capacità di risolverlo rapidamente è un buon indicatore della qualità organizzativa del lavoro del team.

Dalla misurazione alla previsione: definire i SLA

L'introduzione di metriche come quelle viste fin qui, fra l'altro, permette di fare supposizioni sul tipo di performance che potremo aspettarci dal sistema. Quando i numeri iniziano a stabilizzarsi, diventa meno rischioso impegnarsi a rispettare un certo livello di servizio (SLA, **Service Level Agreement**) nei confronti di clienti o stakeholder esterni.

Nel capitolo precedente abbiamo parlato di **classi** di **servizio** delle varie attività, e delle relative politiche di gestione; definire tali policy e mantenerle stabili nel tempo è un requisito per il miglioramento delle prestazioni del sistema. Le metriche possono confermarlo a posteriori.

Le classi di servizio e i corrispondenti SLA

Nei paragrafi precedenti, si era fatto l'esempio delle seguenti classi di servizio:

- **Classe 1: bassa** priorità. Costo del delay non influente. Il trattamento specifico consiste nel mettere in fondo al backlog, lavorare se non c'è altro da fare. Quando la coda delle attività di questa classe supera le 5 unità, mettere in lavorazione la più anziana.
- **Classe 2:** media priorità. Costo del delay varia in base agli accordi contrattuali (rilasci ad ogni iterazione): è significativo ma non enorme. Trattamento specifico: lavorazione urgente ma non bloccante.
- **Classe 3: alta** priorità. Costo del delay: ogni giorno passato in lavorazione costa 1000 €. Trattamento specifico: lavorazione urgente e bloccante; ogni altra attività deve essere interrotta. Se necessario, più persone possono mettersi a lavorare su un task di questo tipo.

Lavorando sulle metriche e analizzando quindi lo stato di salute del sistema, si potrebbero definire opportuni SLA in funzione della probabilità ossia del rischio; per esempio si potrebbero proporre i seguenti livelli di servizio:

- **Classe 1: bassa** – SLA: tempo medio di risoluzione 20 gg; il 90% dei ticket chiusi in 25 gg , tutti entro 30 gg.

- **Classe 2: standard** – SLA: tempo medio di risoluzione 10 gg; il 90% dei ticket chiusi in 15 gg, tutti entro 20 gg.
- **Classe 3: alta** – SLA: tempo medio di risoluzione 5 gg; il 90% dei ticket chiusi in 10 gg, tutti entro 12 gg.

Conclusione

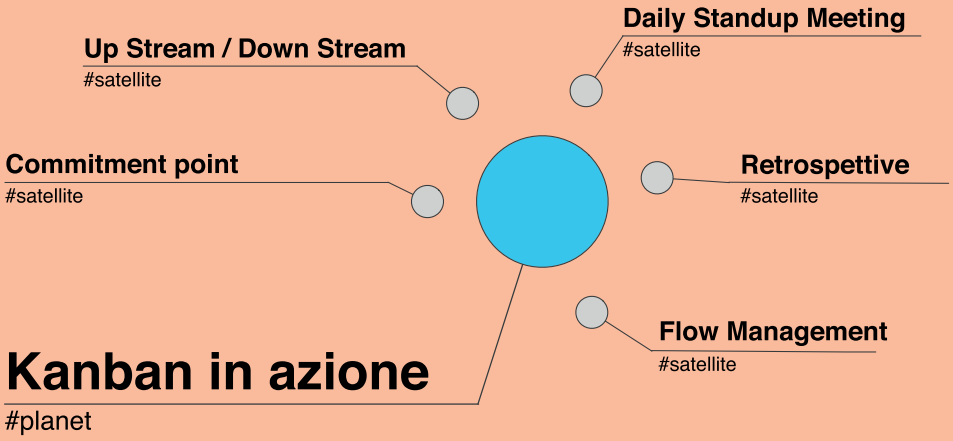
Termina qui la trattazione su misurazioni e metriche da inserire all'interno di un processo Kanban.

Per maggiori approfondimenti sull'uso delle metriche, si rimanda all'appendice di Mattia Battiston — anche se "appendice" potrebbe risultare un termine riduttivo visto che è una trattazione di ottimo livello — che sarà pubblicata nella versione finale completa di questo libro, e i cui argomenti possono essere già trovati in una prima versione nella serie di articoli *Metriche Kanban per il miglioramento continuo* pubblicati nel cosro del 2016 su MokaByte [MET].

Riferimenti

[MET] Mattia Battiston, *Metriche Kanban per il miglioramento continuo*, MokaByte, 2016
http://www.mokabyte.it/2016/02/kanbanmetrics-1/
http://www.mokabyte.it/2016/03/kanbanmetrics-2/
http://www.mokabyte.it/2016/04/kanbanmetrics-3/
http://www.mokabyte.it/2016/05/kanbanmetrics-4/
http://www.mokabyte.it/2016/06/kanbanmetrics-5/
http://www.mokabyte.it/2016/07/kanbanmetrics-6/

Capitolo 5
Kanban in azione: consigli pratici



Lessons learned...

A completamento di questa parte dedicata a Kanban vorrei condividere alcune tecniche e svariati suggerimenti frutto dell'esperienza diretta sul campo: si tratta di suggerimenti utili per risolvere alcuni aspetti concreti come la strategia di condivisione degli allineamenti o come gestione delle retrospettive.

Molte delle idee presenti in questo capitolo sono il frutto di sperimentazioni e tentativi "sul campo" e, pertanto, non è detto che siano adatte a tutti i casi. Se, di fatto, è impossibile trovare "la soluzione" valida per tutte le situazioni operando in un mondo complesso, può comunque essere il caso di condividere quanto raccolto, con la consapevolezza che in alcuni casi potrebbe essere applicabile, in altri no.

Quando si mettono "le cose" nella kanban board?

Una delle questioni più importanti nel processo di gestione di una **kanban board** è il processo di **alimentazione** della stessa. In tal senso può essere utile considerare la lavagna come un sistema di "condotte" nei cui "tubi" le attività scorrono seguendo un flusso che in genere è diviso in due fasi:

- una prima fase di raccolta, alimentazione e relativa selezione di nuove attività;
- una seconda fase di effettiva lavorazione.

Commitment point

Il punto che separa queste due fasi viene in genere definito in inglese **commitment point** dato che rappresenta il passaggio dalla fase in cui le attività sono delle opzioni a quella in cui il team si **impegna** realmente a svolgerle (**committed work**).

Le attività che sono in fase di valutazione, alla sinistra del **commitment point**, sono delle opzioni che potrebbero non passare la selezione. David J. Anderson, in una sua recente pubblicazione su Kanban [EKC], descrive questa parte utilizzando la metafora di un concorso di bellezza, dove le storie fanno a gara per essere scelte: solo quando un item supera il **commitment point** prende importanza; prima di allora, è semplicemente una opzione.

Una volta messa in lavorazione, un'attività non dovrebbe mai essere abbandonata se non in casi eccezionali: per questo motivo si sceglie la parola *commitment*, che indica il "prendersi seriamente un impegno". Dal punto di vista del cliente, la presa in carico di un'attività che passa il commitment point rappresenta una promessa, più o meno esplicita, che tale attività verrà portata a completamento, prima o poi.

Up Stream e Down Stream

Il processo di selezione (prima) e di implementazione (poi) può essere considerato come un unico flusso di lavorazione ("**work flowing like a stream**") che scorre da sinistra verso destra; per questo la zona a sinistra del **commitment point** viene chiamata **Up Stream** (il corso iniziale del flusso), mentre quella alla destra è detta **Down Stream** (la porzione finale del flusso).

Il **lead time**, di cui abbiamo già parlato in passato e di cui parleremo ancora, non dovrebbe tener conto del tempo in cui le attività sono in Up Stream: in questa fase, il tempo di attesa **non** dipende dal processo.

Per valutare e selezionare le attività presenti in Up Stream, si può ricorrere alle tecniche tipiche utilizzate in Product Management o Risk Management. La figura 41 spiega in modo piuttosto chiaro questo schema.

Spesso il **commitment point** è la prima colonna della board sulla quale si impone un WIP limit piuttosto basso.

A volte si impone un **WIP limit** anche in Up Stream per evitare ingolfamenti: inutile prendere in esame nuove idee se il gruppo di lavoro non è in grado di valutare nemmeno quelle già presenti in Up Stream.

Queue replenishment

Al **commitment point** si esegue il cosiddetto **queue replenishment**, vale a dire il "rifornimento della coda": quando ci sono caselle vuote nella prima colonna, bisogna scegliere quali storie meritano di essere tirate (**pull**) dentro il sistema. Il queue replenishment può avvenire **just-in-time**, cioè ogni volta che c'è uno slot vuoto, o essere cadenzato, qualora le persone necessarie per la valutazione/selezione non siano disponibili in modo continuativo e costante.

Il vantaggio della suddivisione in Up Stream e Down Stream

La separazione del processo in queste due fasi offre un interessante strumento di **sense-making**: il modo con cui le attività sono gestite cambia radicalmente se ci si trova a **destra** o a **sinistra** del **commitment point**.

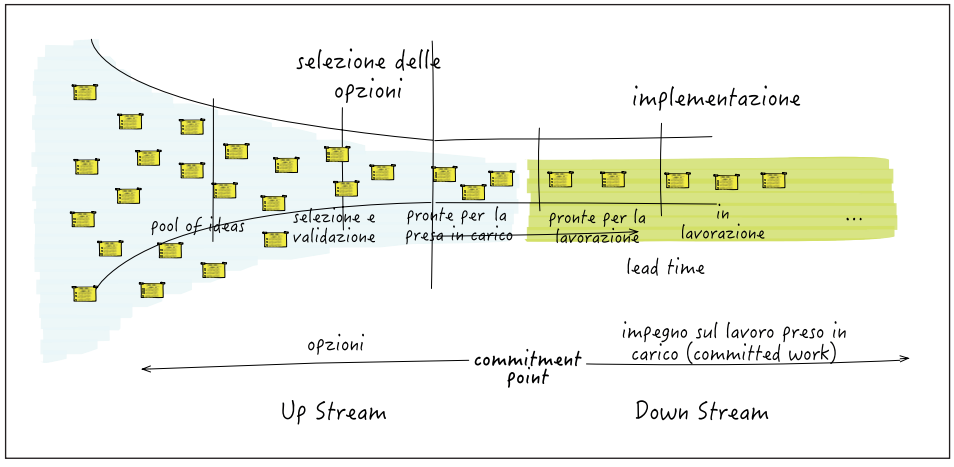


Figura IV.41. Il commitment point separa la parte di selezione delle attività potenziali (Up Stream) da quella della lavorazione delle attività selezionate (Down Stream).

Per esempio il tipo e il significato delle metriche cambia molto a seconda che si consideri l'Up Stream (cono delle opzioni) o il Down Stream (impegno a prendere in carico e completare il lavoro). Durante la fase di lavorazione, il **lead time** assume un significato più concreto e realistico, essendo eliminata ogni indecisione sul "lavorare o non lavorare" un determinato item; questo aiuta a formulare in modo più realistico ipotesi sulle tempistiche di termine dei lavori e di rilascio. In Up Stream, invece, si usano metriche differenti come la lunghezza delle code, il tempo di attesa in determinate sottofasi o addirittura il coefficiente di cancellazione degli item.

Quando si fanno gli standup?

Una delle colonne portanti di Kanban è certamente il modello dei **feedback loops**, dei quali lo stesso Anderson [EKC] ha sottolineato l'importanza:

"Di recente ho adottato un nuovo approccio all'insegnamento del metodo Kanban. Il nuovo corso di Lean Kanban 'Practicing the Kanban Method' si incentra sulle '7 Kanban Cadences', le riunioni tenute ciclicamente che guidano il cambiamento evolutivo e la consegna di servizi 'buoni per lo scopo' per cui sono creati."

Anderson sottolinea che, per implementare i vari **feedback loops**, non necessariamente si deve dar vita a 7 nuovi meeting, ma anzi si potranno interpretare in modo nuovo quelli già esistenti.

Feedback loops e daily standup

Fra i vari **meeting** possibili, certamente il **Daily Standup** preso in prestito da Scrum è uno dei più importanti dal quale si possono trarre maggiori benefici proprio nell'ottica di creare dei feedback.

Il formato tipico di uno **Daily Standup Meeting** di derivazione Scrum prevede che ogni persona elenchi tre elementi fondamentali:

- cosa è stato fatto il giorno prima;
- quali impedimenti si sono incontrati nel fare le cose il giorno precedente;
- cosa ci si appresta a fare il giorno stesso.

Questo approccio ricalca la filosofia tipica di Scrum, in cui ci si orienta fortemente all'analisi delle cose da fare: è un approccio **task-oriented** all'interno dello sprint.

In Kanban invece tipicamente l'attenzione è rivolta verso la **massimizzazione** del **flusso** ed è per questo motivo che può essere utile modificare tale meeting in modo da privilegiare un'analisi **flow-oriented**.

Dal "task-oriented" al "flow-oriented"

Un modo per ottenere questo risultato è quello di abbandonare il classico schema delle tre domande, per passare a una **valutazione** dei **cartellini** presenti sulla **lavagna** da destra a sinistra secondo quello che a volte viene detto stile *walk the board* ("camminare attraverso la lavagna") in modo da implementare un approccio **pull**.

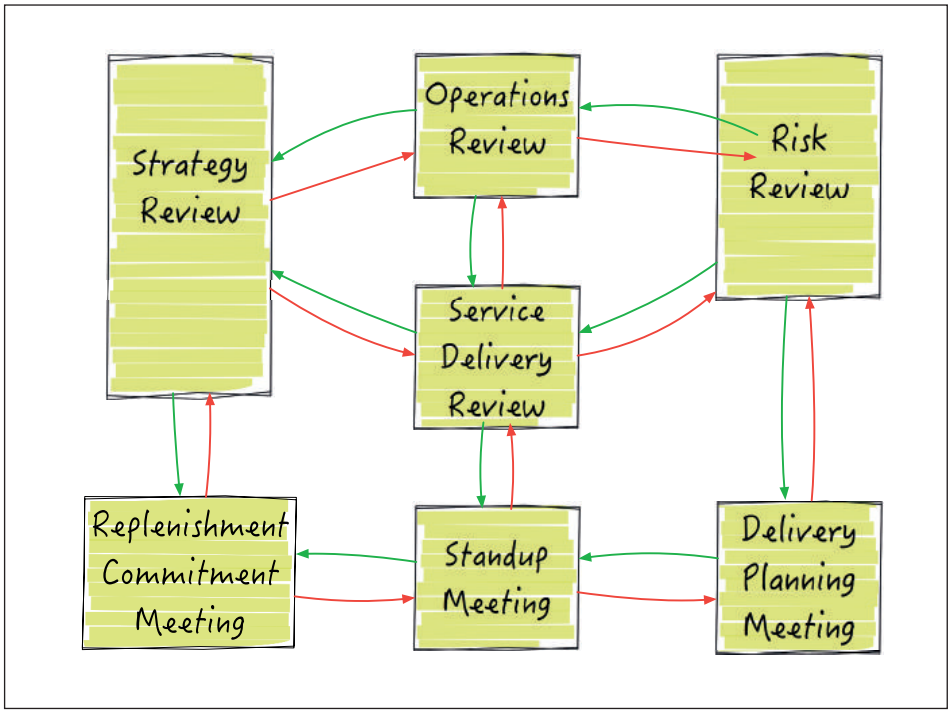


Figura IV.42. Il sistema dei feedback loops di cui parla Anderson può essere visto come un reticolo di interazioni interdipendenti il cui scopo ultimo è il miglioramento delle prestazioni del sistema, ossia ridurre il lead time medio.

Con questa nuova modalità, si segue lo schema riportato di seguito:

- Per prima cosa si analizzano gli eventuali elementi bloccati, cercando di individuare la causa che ne impedisce il completamento. Questi sono senza dubbio i problemi più importanti e conviene risolverli prima possibile: ha senso che vi si dedichi la maggior parte del tempo a disposizione.
- Secondariamente, conviene concentrare l'attenzione sulle attività presenti sulla expedite lane o più in generale sugli emergency items se presenti. È prioritario per il team velocizzare la lavorazione di tali attività, vale a dire risolvere il problema corrispondente, anche a costo di togliere tempo prezioso alla lavorazione di altre attività. In questo caso è utile marcare il cartellino corrispondente all'attività in analisi, per esempio utilizzando un simbolo visibile e comunicativo: un pallino rosso o un altro simbolo di allarme.
- A questo punto si può passare ad analizzare quelle attività che non si sono mosse dall'ultimo standup. Anche se probabilmente è ancora presto per considerarle bloccate, è probabile che siano a rischio. A volte, per il breve tempo intercorso dallo standup precedente, può essere plausibile che non si siano spostate nella board: in tal caso

la discussione non dovrebbe richiedere molto tempo. Qualora invece si dovesse sentire puzza di bruciato, potrà essere utile indagare un po' meglio per capire se si tratta di una situazione normale o se si prefigura un blocco.

- Infine si può passare ad analizzare gli altri elementi rimanenti: in questo caso, se i cartellini sono ancora molti, si può seguire la priorità delle classi di servizio delle attività, partendo da quelle a priorità maggiore (bugs, critical features, altro); aiuta in questi casi la classificazione delle attività per classi di servizio su cui dovrebbe essere definita una priorità.

Durante questo processo di valutazione, per decidere il piano di lavoro quotidiano, sono d'aiuto **dati** e **metriche**, come l'età di cartellino ("Da quanto tempo è in lavorazione questa attività?") o l'effort medio richiesto ("Quanto impiegano solitamente attività di questo tipo per essere completate?").

Vantaggi del daily standup meeting in stile "walk the board"

Seguire la priorità delle attività — si parte dalle attività bloccate per arrivare a quelle delle classi di servizio meno prioritarie — rende il processo di analisi più efficace, perché si agevola la consegna di attività terminate, velocizzando il rilascio di valore all'utente finale. Ma è anche efficiente: se il tempo del meeting termina prima di essere riusciti ad analizzare tutti i cartellini, si saranno trascurate solamente le attività a priorità più bassa, non bloccate, che in definitiva non danno preoccupazioni e che non richiedono quindi particolari discussioni... almeno per ora.

Inoltre, questo stile di conduzione del Daily Standup ha l'effetto di migliorare la collaborazione: alla domanda "Cosa dobbiamo fare oggi per spostare verso destra i cartellini?" in genere si ottiene un coinvolgimento e una partecipazione maggiore da parte di tutte le persone.

Per concludere, è importante notare che il Daily Standup non è altro che uno strumento, per cui può valere la pena di sperimentare o comunque di fare modifiche: si possono sfruttare quei 10 minuti per fare qualcosa di diverso. Per esempio nei gruppi di lavoro di Buffer [BUF] utilizzano lo standup non solo per misurare i **miglioramenti** del **processo** di **lavorazione**, ma anche per analizzare eventuali **miglioramenti personali** dei membri del team, sia per quello che concerne le conoscenze tecniche, che interpersonali.

Flow manager, flow management

Nel paragrafo precedente si è visto come il cambio di approccio nello standup permette di spostare il focus dal fare "la contabilità" dei task da svolgere, per tenere occupate le persone del team, al **miglioramento** del **flusso** di **lavorazione**; a tal proposito è interessante l'introduzione del **ruolo** del **Flow Manager**, una nuova figura che alcuni team stanno iniziando ad adottare con benefici tangibili.

Il **Flow Manager** concentra il suo operato su tutte quelle iniziative volte a velocizzare lo scorrimento dei cartellini sulla board e fra queste rientra anche il guidare un **Daily Standup** con lo stile **walk the board** appena visto.

È un **coach**, per cui **stimola la discussione** all'interno del team, porta all'attenzione delle persone il rispetto o meno delle regole che il team si è date, prima fra tutte il rispetto dei WIP limit. Guida la valutazione delle eccezioni stimolando la creazione di idee innovative.

Il **Flow Manager** può essere visto come una figura parallela allo Scrum Master, con cui ci sono moltissime similitudini: va tenuto presente però che l'attenzione del Flow Manager si concentra principalmente sulla **velocizzazione** del **flusso** e meno sugli aspetti legati alla gestione del progetto [FM].

Quando e come si fa la retrospettiva?

La retrospettiva rappresenta un altro dei **feedback loop** fondamentali: visto che il cuore di Kanban è il **miglioramento continuo**, fare regolarmente delle retrospettive è lo strumento indispensabile per abilitare questo miglioramento.

Retrospettive: differenze tra Scrum e Kanban

Rispetto a Scrum, in Kanban **non** vi è il concetto di **iterazione** che in qualche modo detta il ritmo per l'organizzazione del lavoro, collocando la retrospettiva al termine dello sprint. Per questo motivo, in **Kanban** anche la **frequenza** con cui effettuare le **retrospettive** è **decisa** dal **team**: può essere ogni due settimane, una volta al mese, oppure quando si pensa che sia necessario. Spesso nei team molto maturi c'è la tendenza a non seguire una cadenza regolare, organizzando invece una retrospettiva non appena si pensa che possa essere utile.

Dal punto di vista degli strumenti utilizzabili si può far riferimento alle classiche tecniche di **Agile Retrospective**, utilizzate comunemente dai team agili [RTR]. A causa della popolarità di Scrum, dove la retrospettiva gioca un ruolo fondamentale, la maggior parte delle tecniche e dei formati utilizzati sono quelli adottati dai team che utilizzano Scrum come metodologia di lavoro.

Ciò nonostante, alcuni autori ritengono che passando a Kanban potrebbe essere utile provare a cambiare anche il formato e le tecniche utilizzate per fare le retrospettive: un buon motivo per padroneggiare vari approcci e tecniche diverse per le retrospettive..

Stop the Line Retrospective

Un primo interessante esperimento che alcuni team hanno provato a effettuare prende spunto dal lavoro fatto da Taiichi Ōno in Toyota, introducendo il concetto di **Stop the Line Retrospective** ("retrospettiva con interruzione della linea di produzione"): non appena si verifica un problema "grave" — ovvio che bisogna stabilire prima cosa sia un "problema grave" — il team interrompe il proprio lavoro e si riunisce per capire come risolverlo.

Non sempre questo è un approccio praticabile: capita che il problema sia più complicato di quello che sembra e non basta una riunione di un'ora per trovare la soluzione. Alle volte può capitare che questo approccio non porti ad avere sufficiente

coinvolgimento da parte di tutte le persone del team; il rischio, testimoniato da chi ha provato ad applicare questa tecnica, è che alcune persone si sentano prigioniere, aspettando impazientemente di tornare al proprio lavoro, cosa che di riflesso impatta negativamente anche su chi ha sollevato il problema.

Nella produzione *lean* messa a punto in Toyota questa politica funzionò probabilmente perché Ōno impose che tutti partecipassero in modo attivo, secondo un approccio "imperativo" che oggi valuteremmo poco in linea con i principi dell'agilità: bisogna però considerare il periodo storico e il contesto sociale del Giappone post-bellico. Nel contesto attuale di un team di sviluppo, è ipotizzabile che la cosa possa funzionare meno bene.

Pull the Problem

Una variante meno intrusiva e forse più efficace è quella nota con il nome di **Pull the Problem**, dove il team attacca su una lavagna i problemi incontrati: bug da risolvere, necessità di approfondire determinate tematiche o altro che possa impattare negativamente sul flusso della lavorazione.

Su tale board si impone tipicamente un **limite** al **numero massimo** di problemi: non appena tale limite viene **superato**, il team, al termine del primo standup disponibile, si ferma per un momento di riflessione — una retrospettiva appunto — in cui si cerca di analizzare i problemi. Se il tempo non dovesse bastare, si stabilirà una strategia per approfondire meglio il punto, individuando, ad esempio, una porzione di tempo in un altro momento.

Questa tecnica può funzionare meglio rispetto alla Stop the Line Retrospective vista sopra, perché il "tabellone dei problemi" è, a tutti gli effetti, una Kanban board: senza necessariamente arrivare al superamento del limite, tutto il team si impegna a prendere in lavorazione uno dei cartellini di problema appesi nella board non appena vi è un momento disponibile: per questo si usa il nome **Pull the Problem**.

Operations Review Meeting

Un'altra interessante evoluzione del concetto di retrospettiva è quella che Anderson chiama **Operations Review Meeting**, nota anche con il nome di Metrics Review, che tipicamente viene condotta una volta al mese.

Il team in questo caso si sofferma a rivedere lo stato delle metriche, cercando di evidenziare particolari trend o dati insoliti. Sulla base dell'analisi delle misurazioni effettuate, si discute di come stanno procedendo le cose e delle cause degli attuali trend. Utile in questo caso discutere degli esperimenti in corso, valutando gli eventuali effetti che hanno sul lavoro, sempre tramite l'analisi delle metriche in uso.

Quando si pulisce la colonna del "done"

Questo è un aspetto molto interessante soprattutto perché molti lo considerano un dettaglio: si ripulisce la colonna con le attività "già fatte" quando capita, tanto sono

attività terminate. In linea di principio può essere un approccio corretto, anche se forse potrebbe essere utile capire meglio cosa si intenda per **done**.

Alcuni considerano il termine **done** fuorviante; il team di sviluppo infatti tende a considerare terminata un'attività quando viene rilasciata in produzione o quando non c'è più alcun lavoro da fare; per il cliente invece l'attività è realmente terminata quando la funzione rilasciata risolve il suo problema.

In tal senso potrebbe essere più corretto chiamare l'ultima colonna **released**, e non semplicemente "terminata", e verificare di tanto in tanto se quella attività sta avendo l'effetto desiderato: "OK, è stata rilasciata, ma l'utente la sta utilizzando? È soddisfatto? I suoi problemi sono risolti? Sta avendo l'effetto desiderato in produzione? Sta dando il valore che avevamo stimato?".

A volte è utile separare la colonna del done in due parti: quello che è arrivato **oggi** in **done**, e quello che era **già** in **done**. In questo modo, allo standup successivo, si può avere un'idea più precisa di cosa è stato completato, enfatizzando le attività effettivamente completate dall'ultima riunione. Un piccolo cambiamento di nome che ha un'importante implicazione nella **product ownership** della lavorazione delle attività.

Quante storie far entrare nel sistema

Questo riguarda un po' il "quando" si mettono le cose nella kanban board. Molti team che arrivano da Scrum fanno l'errore di selezionare molte più storie di quante ne finiranno prima del prossimo **replenishment**, e quindi sprecano tempo a discutere dettagli e priorità di storie che neanche inizieranno.

Una semplice tecnica per iniziare è guardare quante storie sono completate solitamente tra un **replenishment** e l'altro, e tirare dentro lo stesso numero di storie. Un passo successivo potrebbe essere quello di iniziare a stimare le storie in base alla complessità o, più correttamente, all'**effort** necessario per il loro completamento.

Riferimenti

[EKC] D. J. Anderson, *Essential Kanban Condensed*
http://www.leankanban.com/guide

[RTR] E. Derby – D. Larsen, *Agile Retrospectives: Making Good Teams Great*, Pragmatic Bookshelf, 2006

[BUF] Buffer
https://goo.gl/cTx9pf

[FM] La figura del Flow Manager
http://goo.gl/HH1mlS

Appendice A
Lover's beer game: una metafora per comprendere il Lean

Introduzione

Con questa appendice, vogliamo parlare del cruciale argomento del **flusso di produzione** che è uno dei punti nodali del **Lean**. Attraverso una metafora cercheremo proprio di comprendere il rapporto tra **flusso degli elementi** in un processo e conseguenze sull'intero sistema.

Ci dedicheremo all'analisi delle **dinamiche** dei **sistemi** di **produzione complessi** e vedremo quali sono gli effetti derivanti da una mancanza di sincronizzazione fra domanda e offerta all'interno del processo di produzione. Le oscillazioni che ne scaturiscono possono portare a una situazione di criticità del sistema, se non addirittura al suo collasso. Il fenomeno alla base di questo esempio è detto **Effetto Forrester** e ha a che fare, fra le altre cose, con la sindrome dei **decisori a razionalità limitata**.

Per presentare il problema, useremo una simulazione nata da un'idea formulata negli anni Sessanta del secolo scorso presso la Sloan School of Management del MIT; si tratta di un **gioco** di **ruolo**, il **Lover's beer game,** in cui gli attori coinvolti devono agire rispettando alcune regole e muovendosi in un ambiente vincolato.

Figura IV.43. Il Beer Game è stato giocato in molte sessioni di serious gaming come strumento per comprendere alcuni concetti basilari della produzione snella (lean).

Giocare per comprendere la realtà

Si riporta in questo paragrafo il risultato di una tipica sessione di gioco; dalle innumerevoli "partite" giocate in tutti questi anni, nell'ambito dei vari corsi di **management** e **lean manifacturing**, è emersa una chiara tendenza: il più delle volte, il risultato finale è stato proprio quello che racconteremo nella storia.

Peter Senge, descrive dettagliatamente questo gioco nel libro *La quinta disciplina* [5TH] e, a tal proposito, ci conferma che il risultato finale **non** dipende dai singoli giocatori, ma dal modo in cui è configurato il **sistema giocatori + ambiente + vincoli**: "non sono le scelte delle singole persone o le cause esterne a condizionare il sistema, ma è il sistema stesso". Interessante notare che il gioco in realtà ha una **corrispondenza** forte con la **realtà**: nella storia recente ci sono stati infatti diversi casi in cui si è potuto assistere a comportamenti sistemici analoghi a quelli descritti in questo capitolo.

Si può prendere per esempio il caso della crisi della produzione dei microchip verificatosi nel 1985, in cui una serie di oscillazioni della domanda/offerta, impattarono sui prezzi e sulla disponibilità dei pezzi, nonché sui ricavi. Una situazione analoga si era verificata un decennio prima nel settore dei semiconduttori, in cui furono coinvolte aziende del calibro di **Siemens** e **Honeywell**. Un altro caso lampante fu quello che si ebbe verso la fine degli anni Ottanta nel settore dell'automobile, dove **General Motors**, **Ford** e **Chrysler** furono vittime di una situazione del tutto analoga.

Per motivi di spazio, la versione del gioco riportata in questo capitolo è necessariamente sintetica. Chi fosse interessato a maggiori dettagli, può senza dubbio far riferimento a *La quinta disciplina* [5TH], libro che è diventato una pietra miliare per la teoria del pensiero sistemico.

La birra Lover's, i tre "attori" e il processo

Questa storia si basa su un prodotto di fantasia, la **Lover's Beer**, una marca di birra inventata ma che faremo finta essere piuttosto popolare fra i giovani di una ipotetica regione degli USA. Questa birra è economica e viene consumata con regolarità ma **senza** raggiungere volumi di vendita delle birre più diffuse. Un bel giorno le vendite aumentano improvvisamente senza un evidente motivo; nel libro si spiega che questa variazione è dovuta all'apparizione della birra nel video musicale di un gruppo piuttosto famoso fra i giovanissimi: per questo tutti i ragazzi in grado di comprare alcolici iniziano a comprarla con più frequenza.

Commerciante, grossista, produttore: tre punti di vista

Gli attori coinvolti in questa storia sono, oltre ai clienti finali, un **commerciante al dettaglio** che vende fra le altre cose la birra Lover's, un **grossista** che rifornisce i negozi di alcolici della zona, fornendo anche la suddetta birra e, infine, la **fabbrica produttrice** della Lover's.

Per spiegare cosa succede in questa simulazione verranno fornite **tre versioni** della stessa storia, dal punto di vista del **commerciante** al dettaglio, del **grossista** e del

produttore. In questa simulazione si immagina che i tre attori **non comunichino** fra loro per telefono o di persona, ma si limitino a scambiarsi ordinativi e consegne tramite un sistema ormai rodato.

Un altro fattore da tenere presente è che, per motivi legati al processo di produzione e di distribuzione, ogni ordinativo viene poi consegnato **dopo 4 settimane**. A causa dell'organizzazione del sistema — un unico fabbricante, un rete di distribuzione limitata e un ampio numero di rivenditori finali — tale ritardo si propaga su tutti i nodi della rete.

Il punto di vista del commerciante

Il **commerciante** al dettaglio **Tom** vende nel suo negozio birre di vario tipo; ma in questo gioco concentriamo l'attenzione su un prodotto in particolare: la **birra Lover's**, prodotto che, forse in virtù del suo prezzo, è particolarmente apprezzato dai più giovani. Per questo motivo Tom sa che regolarmente, settimana dopo settimana, riesce a vendere **4 casse** di questa birra.

Gli ordinativi sono effettuati a inizio settimana e, a causa della regolarità delle vendite, Tom richiede sempre lo stesso quantitativo al grossista inviando un ordine che ormai non ha bisogno di ulteriori spiegazioni.

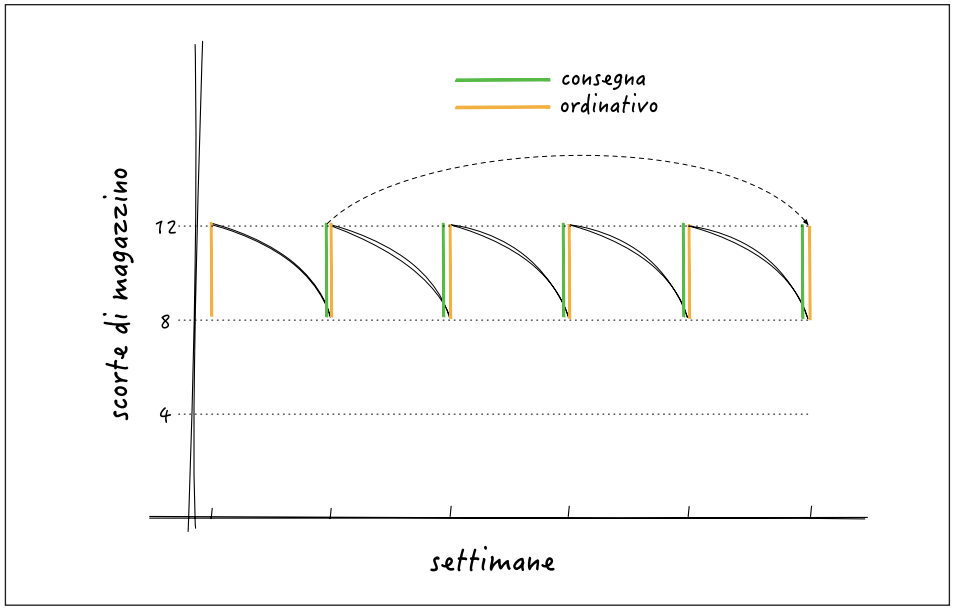


Figura IV.44. Il ciclo di vendita della birra Lover's, misurato al punto vendita al dettaglio, segue un andamento molto regolare; il negoziante, ordinando sempre lo stesso quantitativo di birra, è in grado di asaudire le richieste di tutti i suoi clienti e al contempo di mantenere stabili le scorte di magazzino.

Ordini e consegne

Per semplificare il ragionamento, immaginiamo che le **consegne** della merce e gli **ordinativi** per il futuro avvengano contestualmente, nello stesso istante: per esempio, al lunedì, il camion del grossista consegna a Tom le 4 casse di birra, e il commerciante consegna all'autista un foglio su cui è scritto l'ordine successivo.

Tom non conosce quindi il grossista, né tantomeno conosce i dettagli di chi e come produca la birra, fattore che, come potremo vedere fra un po', si rivelerà fondamentale... in negativo.

Egli sa che ogni ordine viene consegnato dopo 4 settimane, ma non se ne preoccupa, sia per la regolarità con cui riesce a vendere tale birra, sia perché tiene in magazzino una piccola scorta aggiuntiva, in modo che, ad ogni inizio settimana, abbia sempre 12 casse di questo prodotto. Quindi la scorta oscilla fra le 8 e le 12 casse, cosicché egli possa mantenere sempre un certo margine di sicurezza.

Questo accade regolarmente da molto tempo dando luogo a un andamento altalenante ma piuttosto ciclico e in media costante, che è rappresentabile con il grafico riportato in figura 44.

L'andamento delle vendite e dei rifornimenti, settimana dopo settimana

Prendiamo in esame una settimana qualsiasi di questo periodo in cui il comportamento è stabile, settimana che chiameremo convenzionalmente **settimana n. 1**: in questa settimana il commerciante viene rifornito dello stesso quantitativo di birra che ha ha venduto la settimana precedente.

Poi improvvisamente, **le cose cambiano**: senza un motivo apparente, al termine della settimana successiva, la **settimana n. 2**, le vendite aumentano raddoppiando da **4** casse a **8** casse. Il venditore non se ne preoccupa più di tanto, visto che la variazione, pur sostanziosa in percentuale, è comunque un evento isolato... o almeno così crede: forse è stato più caldo del solito oppure forse qualcuno ha dato una festa. Inoltre le scorte che Tom ha in magazzino (8 casse oltre alle 4 che oscillano in entrata e uscita) gli permettono di "assorbire" questo sbalzo.

Ciò nonostante, si prepara ad aumentare l'ordinativo il lunedì successivo — quello che dà inizio alla **settimana n. 3** — in modo da pareggiare l'aumento delle vendite riportando il magazzino a 12 casse. Purtroppo all'inizio della **settimana n. 3** il rifornimento che arriva è di sole 4 casse: non dimentichiamo infatti che **ogni consegna** si riferisce all'**ordine** fatto **4 settimane prima**.

Pertanto, complessivamente, il magazzino è sceso da 12 a 8 casse. Se la variazione nelle vendite è frutto di un caso isolato — qualche festa in più, il caldo dell'estate o chissà cos'altro — Tom sa che nell'arco di 4 settimane il suo magazzino tornerà ad avere una scorta di 12 casse: in fondo, la scorta serviva proprio per assorbire eventuali oscillazioni.

Per il momento Tom quindi non si preoccupa particolarmente e continua a comportarsi in **maniera razionale** sulla **base** della **propria esperienza**.

Cambiamenti decisivi

Durante la **settimana n. 3**, il commerciante comprende che quanto accaduto non era un caso isolato, ma che anzi le vendite continuano su un ritmo di 8 casse a settimana. Essendoci in magazzino solamente 8 casse, al termine della terza settimana il negoziante si trova senza birra.

Alla consegna del lunedì della **settimana n. 4**, riceverà solamente 4 casse di birra: l'ordinativo maggiorato è stato fatto solamente alla settimana n. 3 per cui le 8 casse arriveranno alla settimana n. 7.

Deve quindi risolvere rapidamente il problema e sa che probabilmente a metà settimana si troverà senza birra, andando quindi in debito di merce; per esempio potrebbe segnarsi il nome dei clienti e richiamare quando la merce sarà nuovamente disponibile.

Per questo, nemmeno le 8 casse ordinate la settimana precedente possono bastare: adesso non solo deve pareggiare lo svuotamento del magazzino ma anche fronteggiare il "debito" che si creerà fra pochi giorni. Per questo, decide di **agire di anticipo** e aumenta l'ordinativo a 12 casse.

Proseguendo questo trend, settimana dopo settimana Tom si trova ad aumentare sempre più il proprio ordinativo fino alla settimana 7: adesso il trend delle consegne dovrebbe invertirsi, dato che all'inizio di **settimana n. 7** dovrebbe finalmente arrivare la prima consegna maggiorata (8 casse).

Scarsità di merce

In realtà, al lunedì di **settimana n. 7**, il grossista consegna solo 5 casse rispetto alle 8 ordinate. E nelle settimane successive le cose non cambiano; anzi addirittura peggiorano. Completamente in balia di fattori indipendenti dalla sua volontà, il negoziante non può far altro che arrendersi all'evidenza dei fatti: **senza conoscere** il **motivo** della crescita della domanda né il motivo della scarsità delle consegne**, non** riuscirà a soddisfare le richieste dei propri clienti. Il problema principale è che la mancanza di birra Lover's ha rallentato la vendita di merce di contorno normalmente acquistata dai clienti che comprano la Lover's.

Il punto di vista del grossista

Susan è la responsabile degli ordini e delle vendite di un **grossista** di zona. Nel suo ruolo, Susan è testimone di un andamento analogo a quello che ha sperimentato Tom, il negoziante, anche se ovviamente con volumi certamente maggiori e su un altro tipo di scala: Susan è il riferimento per il rifornimento di **molti negozi** della sua zona, piccoli supermercati o genericamente intere zone molto ampie ma a bassa densità di popolazione.

Se un negoziante ragiona per le consegne in termini di casse di birra, l'unità di misura di Susan sono i **bancali** che sono movimentati tramite camion verso i vari negozianti della zona coperta dalla propria zona. Come il negoziante finale, anche Susan ha un **tempo di attesa di 4 settimane** per ricevere la birra che ordina alla fabbrica.

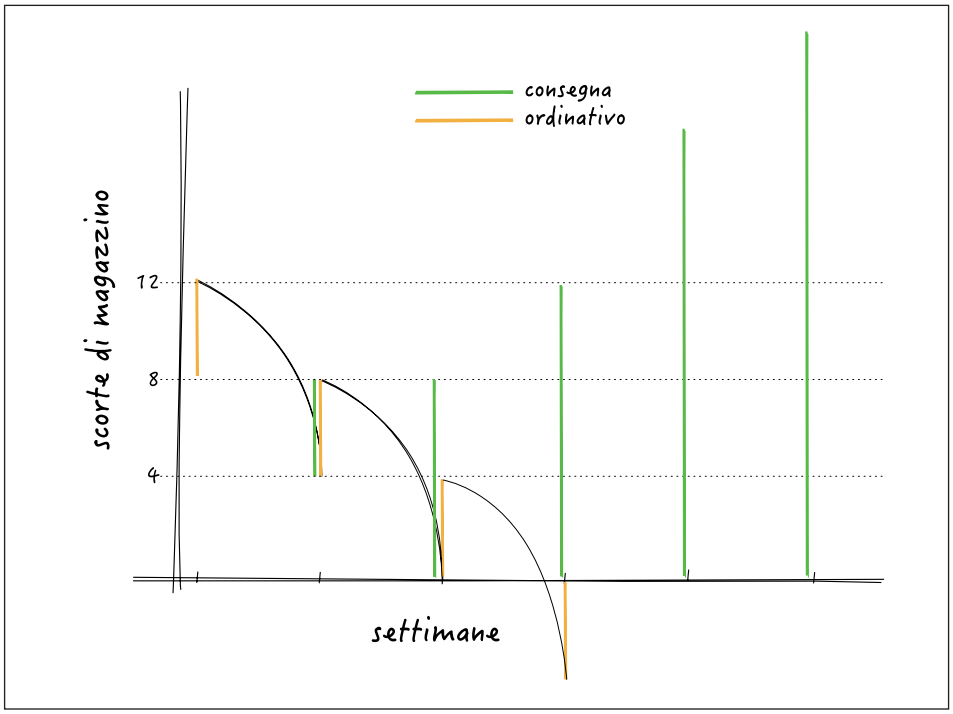


Figura IV.45. In questo grafico è riportato l'andamento delle vendite al negozio: l'alterazione nella domanda consuma in breve tempo le scorte del negozio e obbliga il negoziante a impegnarsi con i clienti più affenzionati a consegnare la birra non appena questa arriverà in negozio. Per questo il grafico scende sotto il livello dello zero.

L'andamento delle vendite del grossista

Anche Susan riscontra un aumento delle vendite alla **settimana n. 4**. A dire il vero aveva avuto una piccola **oscillazione** alla **settimana n. 3**, ma anche nel suo caso aveva ipotizzato si trattasse di una variazione più o meno casuale.

Il **grossista**, per la sua posizione all'interno della rete di vendita, di fatto **concentra** e **somma** l'**andamento** delle vendite dei **negozianti** che dipendono da lui. Per questo l'oscillazione che Susan nota alla terza settimana è quasi impercettibile, mentre quella della quarta è sostanziale e maggiore di qualche ordine di grandezza.

Il **grossista**, non appena si rende conto che il nuovo trend non è una semplice oscillazione, interviene modificando gli ordini: anche in questo caso interviene prima con un semplice incremento poi con maggiorazioni più sostanziose.

Susan pensa che se la variazione delle vendite non sarà troppo repentina, le **scorte** di magazzino dovrebbero essere sufficienti per assorbire l'aumento delle richieste in attesa che la fabbrica consegni gli ordinativi aumentati, vale a dire **dopo dopo 4 settimane**.

Scorte di magazzino insufficienti

Purtroppo per Susan, l'aumento di domanda da parte dei commercianti al dettaglio non si arresta, ma anzi aumenta man mano che passa il tempo. Se inizialmente la variazione poteva essere assorbita con le scorte di magazzino o comunque ridistribuendo le consegne ai vari negozianti, dopo qualche tempo arrivano alla dimensione di criticità massima.

Susan, con qualche settimana di ritardo rispetto al negoziante, sperimenta la sua criticità: non appena si rende conto che le nuove richieste stanno erodendo le scorte, reagisce prima razionalmente, aumentando quel tanto per colmare lo svuotamento del **magazzino**; successivamente, trovandosi in difficoltà, prova forzare la mano con la fabbrica aumentando in maniera "sostanziosa" gli ordinativi alla fabbrica.

Anche il **grossista** quindi inizia ad assumere un atteggiamento **emotivo**, guidato dall'**ansia** di non riuscire a evadere tutti gli ordini: i **commercianti** continuano a chiedere sempre più birra e il **grossista** non ne ha. Come avremo modo di vedere, la fabbrica aumenterà la produzione fra qualche settimana e i risultati si vedranno solo fra qualche tempo. Nel frattempo che fare?

Arrivano i nostri...

Verso la **settimana n. 16**, la **fabbrica**, che, in concomitanza dell'aumento delle richieste da pare dei grossisti, aveva deciso di potenziare le proprie strutture e di produrre più birra, inizia a rispondere all'aumento degli ordinativi consegnando più merce di prima, riuscendo in questo modo a coprire quasi tutti gli ordini fatti dai **grossisti** 4 settimane prima.

Susan inizia a essere fiduciosa: la merce appena ricevuta non è ancora sufficiente a coprire il debito di consegne, ma è rassicurata dall'inversione di tendenza da parte della fabbrica e si attende quindi che le prossime consegne potranno colmare tale carenza. I suoi ordinativi infatti sono aumentati in maniera esponenziale.

Inversione di tendenza

Contemporaneamente, dal canale di vendita si nota una leggera **flessione** negli ordinativi. Susan non se ne cura, pensa che molto probabilmente i negozianti, che avevano più o meno seguito lo stesso trend negli ordinativi, e avevano ecceduto un po' nella settimana precedente, adesso rallentano un po'.

In questa fase, con il produttore che inizia ad aumentare la produzione e il magazzino che finalmente inizia a riempirsi nuovamente, il rallentamento degli ordinativi da parte dei negozianti è visto come un fatto positivo. In fondo Susan ha ancora un debito considerevole con i negozianti.

Nelle settimane successive le consegne da parte della fabbrica iniziano ad essere regolari e rispettare quanto richiesto al momento dell'ordine **quattro settimane prima**, quando Susan aveva aumentato gli ordinativi del doppio e poi del triplo rispetto alla **media storica**.

Ora che le cose iniziano a mettersi al meglio, accade una cosa impensabile fino a poco prima: purtroppo, gli ordinativi da parte dei **commercianti** si bloccano del tutto. Nessun venditore finale sembra aver più bisogno di birra Lover's!

Lo stato delle cose

La cosa drammatica è che, nelle settimane seguenti, le consegne da parte del produttore, sempre in accordo con quanto richiesto 4 settimane prima, aumentano esponenzialmente. Adesso stanno consegnando 10 se non 20 volte rispetto alla **media storica**. La **merce** si **accumula** nel **magazzino** mentre i negozianti continuano a non ordinare nulla.

Susan si rende conto che, con il trend attuale, se anche i negozianti dovessero riprendere a ordinare birra Lover's, ci vorrà molto tempo per poter **smaltire** tutta quella merce che si sta accumulando sui bancali in magazzino. Probabilmente non ci riuscirà e probabilmente è a rischio la sopravvivenza della sua azienda.

Il tutto senza che il **grossista** abbia compreso il reale motivo di quanto accaduto. Ad oggi non ha ancora avuto modo di parlare né con qualcuno dei **commercianti** né con un rappresentante della **fabbrica** della Lover's Beer; è stata sempre troppo occupata a gestire gli sbalzi nella lavorazione. La sola cosa che può fare per le prossime settimane è bloccare completamente ogni ordinativo per l'acquisto di altra birra.

Il punto di vista del produttore

Presso la **fabbrica** di produzione della birra Lover's, l'andamento delle vendite segue un comportamento molto simile a quello che abbiamo visto presso il **grossista** e i singoli **rivenditori finali**, anche se ovviamente con volumi notevolmente maggiori.

Dopo una piccola variazione alle prime settimane, le richieste hanno registrato un sensibile **aumento**, con un trend che in poco tempo ha messo in difficoltà il reparto produttivo che non è riuscito a rimanere in pari con le sempre maggiori richieste. Analogamente, anche in questo caso, dopo questo picco di ordinativi, le richieste hanno subito un brusco **rallentamento**, per arrivare alla totale interruzione di ogni forma di ordinativo.

Le variazioni negli ordinativi

Il primo aumento negli ordinativi per i volumi della fabbrica non viene realmente percepito come **variazione della domanda**, ma come una **perturbazione circoscritta**. Anche in questo caso i responsabili alle vendite e alla produzione non hanno preso alcuna iniziativa, immaginando che l'alterazione potesse tranquillamente essere assorbita con le scorte di magazzino che sono sempre disponibili presso la fabbrica.

Quando successivamente si è visto che il trend degli ordinativi rimaneva in costante crescita, i responsabili delle vendite hanno compreso che non si trattava di una variazione isolata, ma di qualcosa di più strutturato.

In quell'istante si è provato a tamponare con un intervento sulle modalità di consegna, cercando di ottimizzare le poche scorte di merce disponibile, per esempio

provando a soddisfare i distributori più importanti o mandando solo parte della merce ordinata. Contemporaneamente i responsabili della produzione si sono attivati per intervenire strutturalmente: prima assumendo nuovi dipendenti, poi provvedendo a potenziare i macchinari per realizzare volumi maggiori di birra.

Purtroppo, prima che le nuove attrezzatura potessero entrare in funzione e potessero consentire un sostanziale incremento della produzione, gli ordinativi si sono interrotti di colpo. Tutto a un tratto nessuno voleva più birra.

Cosa era successo? Oltre al rammarico di non essere riusciti a soddisfare una notevole quantità di richieste (mancati profitti) la fabbrica si ritrova adesso con dei costi strutturali notevolmente aumentati (nuovi assunti e attrezzature da pagare). Per la fabbrica, se gli ordinativi non fossero tornati ai volumi registrati prima dell'interruzione, tutto ciò avrebbe potuto significare un grosso problema.

Il punto di vista del responsabile del marketing e vendite della fabbrica

Nel libro di Senge, il racconto prosegue con l'aggiunta di un'**appendice** in cui si narra del punto di vista di **Marc**, il **responsabile del marketing e delle vendite** della fabbrica. Marc è stato assunto di recente all'interno dell'azienda con lo scopo di far crescere le vendite e si ritrova a vivere in pieno tutta l'oscillazione: da un incremento incontrollato al successivo crollo.

Quando Marc vede che la birra inizia ad accumularsi nel magazzino, nel tentare di **comprenderne** le **cause** e trovare un rimedio, intraprende un viaggio — purtroppo tardivo — presso la rete di vendita andando a parlare con Susan, la responsabile di uno dei **grossisti** che distribuisce la birra Lover's. La conversazione che si instaura mette in evidenza in modo piuttosto chiaro alcune possibile cause di quanto è successo.

Il confronto tra fabbrica e grossisti

Susan accoglie Marc, mostrando le scorte all'interno del proprio magazzino, scorte che si sono accumulate non appena dalla fabbrica hanno iniziato a consegnare gli **ordinativi** maggiorati. Marc frustrato e arrabbiato si interroga a voce alta su come tutto questo sia stato possibile...

Susan, cercando di immaginare quanto tempo ci vorrà per smaltire tutte quelle scorte, condivide la frustrazione e la rabbia di Marc. Entrambi sono frastornati e preoccupati della situazione.

Marc esordisce: "È una tragedia... ma come è potuto succedere?".

Susan, dimostrando anche in questo caso una visione locale, perché non pensa di avere alcuna responsabilità, molto semplicemente ribatte: "Non è colpa nostra; sicuramente i commercianti al dettaglio non si sono saputi organizzare o forse non sanno gestire il mercato".

Marc, nella sua testa, dà la colpa ai rivenditori, accomunando, in preda a un momento di sconforto emotivo, **commercianti** al dettaglio e **grossisti**. Certamente quanto

accaduto è l'ennesima dimostrazione della variabilità del mercato e dell'incostanza dei consumatori. I due si congedano senza avere alcuna risposta per risolvere il problema, ma promettendo di risentirsi nei giorni successivi per provare a condividere qualche strategia.

Il confronto tra frabbrica e commercianti al dettaglio

Nel viaggio di rientro verso il suo ufficio, incidentalmente **Marc** si trova a passare davanti a un negozio che vende, fra le altre cose, la **Birra Lover's**. Per la prima volta da quando lavora alla Lover's, entra nel negozio di un **commerciante al dettaglio** per parlare con il titolare, **Andrew**.

Dopo le presentazioni di rito, Andrew porta Marc nel retro del proprio negozio per mostrare le giacenze di magazzino e anche in questo caso la scena, con le dovute proporzioni, è simile: il retro del negozio è pieno di casse di birra Lover's. Con il trend di vendita attuale, per vendere tutte quelle casse di birra ci vorranno molte settimane. Marc rimane pietrificato: se tutti i negozianti della zona e delle aree vicine sono nella medesima situazione, la fabbrica corre grossi rischi per la propria sopravvivenza, perché non dovrà produrre più nulla per mesi.

Marc chiede i motivi di questo stallo improvviso; il **commerciante** serenamente risponde: "Non saprei, non è colpa nostra. Dopo che è uscito quel video musicale in cui appariva la Lover's, abbiamo avuto un incremento nelle richieste dai nostri clienti; da 4 casse di media, siamo passati a 8 nel giro di una settimana..."

Marc interrompe: "Va bene. Prima le vendite sono esplose... Ma come mai poi sono crollate?"

Andrew ribatte: "No, attenzione! Le **vendite non sono né esplose né crollate**; noi adesso continuiamo a vendere 8 casse di birra a settimana rispetto alle 4 iniziali. Il problema è che voi non ci stavate mandando la birra che ci serviva e stavamo accumulando ritardi e per mantenere i nostri clienti abbiamo dovuto aumentare gli ordinativi".

Marc: "Ma noi abbiamo consegnato gli ordinativi non appena ci è stato possibile!"

Andrew: "Non so che dire... Forse il grossista ha sbagliato qualcosa, tant'è che vorremmo cambiarlo. Di fatto abbiamo ricevuto la birra con ritardo, addirittura in un certo momento, nonostante noi aumentassimo le richieste, le consegne arrivavano incomplete, in misura molto minore rispetto all'ordine. Quando poi finalmente avete iniziato a consegnare la merce, ci avete mandato prima la giusta quantità, poi molto più del necessario. A quel punto abbiamo dovuto interrompere gli ordini".

La storia fin qui

Ricapitoliamo brevemente la sequenza di eventi che si sono susseguiti:

- Una piccola oscillazione nella domanda da parte dei consumatori finali viene prima sottovalutata dalla rete di vendita, poi presa in seria considerazione dal negoziante, che prima sopperisce con le scorte di magazzino, poi provvede ad aumentare gli ordinativi al grossista.

- Le consegne avvengono con 4 settimane di "ritardo" dalla effettiva creazione dell'ordine: per 4 settimane quindi il negoziante non riceve alcun incremento della merce consegnata.
- Il negoziante, prima segue un approccio razionale; quando invece le scorte non bastano più, e non è in grado di soddisfare la sua clientela più affezionata, inizia a reagire in maniera impulsiva aumentando gli ordinativi in misura maggiore di quella realmente necessario. Confonde l'aumento della richiesta settimanale, con le richieste passate non soddisfatte, come se tutti i clienti che non hanno comprato nei giorni in cui la birra non era disponibile, si ripromettessero poi di comprare anche gli "arretrati".
- Questo schema si verifica con lo stesso andamento presso gli altri nodi della filiera (distribuzione e produzione), cosa che porta al blocco delle consegne: la fabbrica non era pronta all'aumento improvviso delle richieste.
- Non essendo disponibile prodotto già presso la fabbrica, i venditori finali dopo le 4 settimane canoniche cominciano a ricevere addirittura meno scorte di quelle ordinate. A questo, reagiscono in modo ancora più irrazionale aumentando le richieste.
- Finalmente il processo si sblocca, perché la fabbrica si attrezza per moltiplicare la sua capacità produttiva. Ma a questo punto, le richieste da parte di venditori finali e distribuzione si bloccheranno, perché entrambi si son visti recapitare un quantitativo di merce enormemente maggiore rispetto alle necessità.

Ancora una volta è necessario sottolineare che la variazione della domanda finale era minima e che nessun cliente probabilmente avrebbe poi comprato tutta la birra che non aveva trovato a scaffale nelle settimane precedenti.

Il processo quindi si traduce in una specie di **onda** fra domanda e offerta, che ha messo in difficoltà tutti i nodi della filiera: il negoziante prima non ha saputo soddisfare la richiesta dei clienti; poi si è ritrovato nel magazzino un'eccedenza di birra che per essere smaltita avrebbe richiesto molte settimane; il distributore ha subìto una sorte analoga e quindi, come il negoziante, ha interrotto gli ordinativi quando si è trovato nel magazzino bancali pieni di birra.

In fabbrica inizialmente non sono stati in grado di evadere l'incremento degli ordinativi ma non hanno subito alcun danno economico diretto, se non mancanza di un ricavo concreto; poi, quando hanno fatto gli investimenti necessari per incrementare la produzione, hanno subìto l'arresto della domanda, con conseguente impossibilità di ripagare gli investimenti fatti, oltre anche in questo caso ai magazzini pieni.

Alcune considerazioni

Anche se la storia nel libro di Senge prosegue ancora, noi ci interrompiamo in questo punto, dato che le cose viste mettono in evidenza già alcuni fattori molto interessanti. Questa storiella ha, ovviamente, tutti i limiti di una storia "didattica" e con scopi esemplificativi. Da una analisi distaccata della storia appare piuttosto chiaro dove sia il problema; ma perché è accaduto? Perché **ogni attore** della **filiera** ha commesso un errore

così banale agendo senza **tenere in considerazione le tempistica di consegna** e non ha saputo valutare gli aspetti **logistici** e le **conseguenze** delle proprie azioni?

Per provare a rispondere a queste domande, è forse utile analizzare la storia sia da un punto di vista **sistemico**, come fa Senge nel suo libro, sia da un punto di vista **organizzativo** e di **processo**, ossia applicando alcune delle considerazioni fatte in precedenza.

Considerazioni sistemiche

Perter Senge fornisce tre massime che emergono dalla storia appena raccontata:

- la struttura influenza il comportamento;
- la struttura dei sistemi umani è "sottile";
- la soluzione può arrivare da nuovi modi di pensare.

La struttura influenza il comportamento

Studi e analisi in svariati settori hanno dimostrato che **differenti persone** hanno prodotto **risultati simili** se messe a lavorare nello **stesso contesto** o in contesti simili.

Quando ci sono cali di performance o problemi di una qualche natura, la prima tendenza è quella di cercare qualcuno o qualcosa su cui far **ricadere la responsabilità**. In realtà le cause delle decisioni che prendiamo nelle nostre iniziative all'interno di un'organizzazione sono frutto dell'organizzazione stessa; quasi mai la responsabilità è in capo a un singolo o a qualcosa di esterno, ma ricade in gran parte sul **sistema organizzazione**, in cui ricomprendiamo le persone, le regole, i processi, i flussi informativi, gli stakeholder, le risorse. Probabilmente ognuno di noi, al posto di Tom si sarebbe comportato nello stesso modo.

La struttura dei sistemi umani è "sottile"

Siamo portati a pensare al concetto di struttura come un insieme di vincoli che agiscono sugli individui. Ma la struttura nei **sistemi complessi** può essere definita sulla base delle **sofisticate interrelazioni** che si creano al suo interno e che ne regolano il comportamento finale.

In un'azienda, come nell'esempio della birra Lover's, il risultato del sistema di vendita, dal consumatore al produttore, è dato dalla complessità **risultante** dalle azioni e decisioni dei vari attori che si muovono all'interno del sistema. Nessuno di questi può pilotare il sistema, ma eventualmente influenzarne il comportamento che sarà comunque il risultato dell'azione di tutti.

La soluzione può arrivare da nuovi modi di pensare

Quando ci troviamo ad agire per affrontare dei problemi all'interno di **organizzazioni complesse**, pur avendo le risorse e le capacità per **risolvere** determinare situazioni, non le esercitiamo perché troppo presi dal focalizzare la nostra attenzione sulle nostre decisioni, senza renderci conto che le nostre azioni interferiscono con quelle degli altri.

Nella storia della birra Lover's, i **singoli attori** osservano e si concentrano **solo** sul **proprio operato** cercando una **soluzione locale** che abbia ricadute immediate nel **proprio ambito**, senza invece considerare l'**intero** sistema in cui tutti agiscono e si influenzano.

E quindi?

Avere quindi una visione sistemica, come proposto da Peter Senge, è di fondamentale importanza per interpretare il comportamento degli attori nella simulazione della Lover's Beer.

Di seguito, proveremo a comprendere come sia possibile spiegare quanto successo nella storiella della Birra Lover's grazie agli strumenti tipici dell'agilità, applicando i concetti visti in precedenza: **Lean Production**, **Cycle Time**, **Kanban**, **metriche** e altro ancora.

Arricchiremo il nostro bagaglio di conoscenze introducendo un interessante fenomeno, detto **effetto Forrester** (o "effetto frusta" o *bullwhip effect*) e vedendo come questo sia causato fra le altre cose dai cosiddetti **decisori a razionalità limitata** e dalla mancanza di alcuni requisiti che sono ritenuti essenziali all'interno di un progetto agile.

Effetto Forrester

Da un punto di vista teorico, il comportamento del sistema preso in esame e le conseguenze delle azioni intraprese dai vari attori, sono spiegabili mediante il cosiddetto **rffetto Forrester**, detto anche **effetto frusta** o **Bullwhip**.

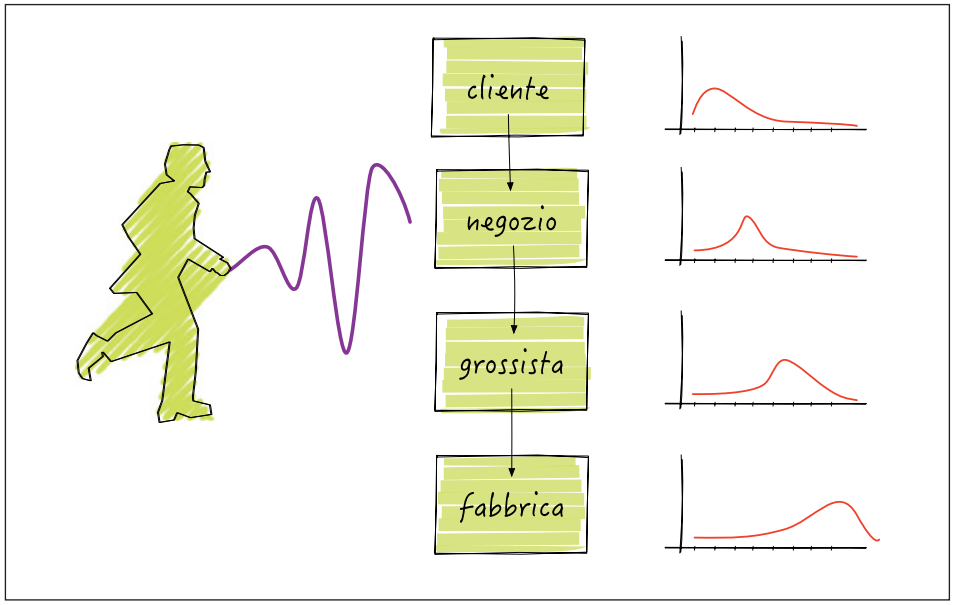


Figura IV.46. L'effetto Forrester o effetto "frusta".

In determinate condizioni, tale fenomeno si manifesta all'interno di una catena di produzione, tramite una oscillazione della domanda e dell'offerta portando alla formazione di una specie di onda che, proprio come la corda di una frusta, si propaga **amplificandosi** man mano che ci si allontana dal mercato finale e si risale la catena di produzione. È tipicamente applicabile ai sistemi di catena di fornitura ma, come vedremo, lo possiamo vedere in azione anche all'interno di un sistema IT come quello della "fabbrica" del software.

Cause principali dell'effetto "frusta"

Probabilmente la causa principale che innesca un effetto frusta è la presenza di attori che prendono decisioni sulla base di **informazioni circoscritte** solo al loro ambito di intervento e che, per questo, vengono chiamati decisori a "**razionalità limitata**": essi non hanno informazioni dalla periferia rispetto al loro contesto d'azione, ossia non hanno sotto controllo tutto il sistema nel suo complesso.

In questo contesto, maggiore è la **lunghezza** della catena di produzione/informazione, maggiore sarà l'**inefficienza** del sistema complessivo; più precisamente tanto più ci si allontana dal cliente finale — o più genericamente dal punto di verifica — maggiore sarà l'errore delle informazioni e quindi il rischio derivante dalle azioni intraprese. Per questo si parla di **onda**, oscillazione o genericamente di **effetto frusta** (figura 46).

Filiera lunga

In agilità, quando si deve sviluppare un prodotto software, o genericamente un prodotto IT, la lunghezza, o meglio la cortezza, della filiera di produzione è un aspetto preso in seria considerazione, visto che è noto a cosa possa portare una catena lunga.

In Scrum per esempio si cerca di ridurre il numero di intermediari organizzando team in cui tutti gli attori si parlano, collaborano e decidono. Per questo si cerca di eliminare gli **intermediari**, di mettere in **comunicazione direttamente** il team con il cliente/utente e il team con il management.

È per questo motivo che un Product Owner che faccia da intermediario fra il cliente e il team di sviluppo incorre nel rischio di allungare la catena.

Tempi di lavorazione

Altro fattore che impatta negativamente è il **tempo di lavorazione** del prodotto: come abbiamo avuto modo di vedere in modo dettagliato nei capitoli precedenti, il tempo di lavorazione, o **Lead Time**, impatta sulle prestazioni complessive del sistema.

In questo caso, è causa — o effetto, dipende dai punti di vista — delle decisioni prese da un lato o dall'altro della catena: se il negoziante avesse potuto avere la sua birra in tempi più rapidi, avrebbe potuto prendere le proprie decisioni in maniera più coerente con la realtà delle cose. Analogamente, se la fabbrica avesse evaso gli ordini in tempi minori, avrebbe potuto recepire i segnali dal mercato in modo più realistico e magari prendere per tempo le adeguate contromisure.

Latenza

Lunghezza della **catena** ed elevato **Lead Time** sono due fattori che ci portano a considerare le cose da un altro punto di vista, quello della **latenza** nella **risposta** del **sistema** alle **sollecitazioni esterne**.

Questo fenomeno si può spiegare con un paio di esempi piuttosto semplici: uno che probabilmente molti lettori possono aver sperimentato direttamente; l'altro solo se si è partecipato a un corso per imparare a pilotare un aereo.

Il primo è quello che si sperimenta quando per esempio si prova a guidare una bicicletta o una motocicletta con il tubo canotto dello sterzo che non si muove con la dovuta libertà (serraggio troppo alto o qualche cuscinetto bloccato). Per guidare una bici normalmente si apportano delle piccole e costanti correzioni alla traiettoria per rimanere in equilibrio. In questo caso il manubrio risponde in maniera dura o addirittura a scatti: la guida non è armonica ma anzi scattosa. Il ciclista inizia con delle piccole oscillazioni per finire con degli ondeggiamenti più ampi, che portano a volte a cadere.

Un effetto simile capita a chi pilota un aereo: per correggere l'assetto egli tira la cloche per alzare il muso. Ma l'effetto della azione non è immediato, per cui il pilota inesperto tira la cloche più del dovuto; dopo poco il muso dell'aereo si alza più di quanto necessario, e per questo il pilota dà una correzione vistosa verso il basso. Ma siccome anche in questo caso l'effetto non è immediato, il pilota alle prime armi comincia ad avere un po' di ansia… e abbassa troppo. Quando il muso si abbassa, allora tira di nuovo la cloche... e la tira troppo.

Di fatto in quel caso il tipo di risposta del sistema non può essere modificata, per cui pare che nell'addestramento ai piloti venga insegnato proprio ad evitare questo loop pernicioso, apportando piccole modifiche e riportando sempre la cloche in assetto.

Il tema della **latenza** nella risposta di un sistema è uno degli aspetti più importanti su cui si fonda buona parte della filosofia **agile**: seguire **iterazioni brevi** in Scrum, organizzare **frequenti incontri** con il cliente/utente per fare delle demo, raccogliere il maggior numero di **feedback** possibili sono tutte azioni che vanno nella direzione di aumentare la **reattività** del sistema, ossia intraprendere decisioni i cui effetti siano verificabili prima possibile.

Alcune cause dirette dell'effetto "frusta"

Accorciare la filiera, ridurre il lead time, aumentare la reattività sono tre strategie efficaci per limitare gli effetti negativi della **frusta**. Esistono poi anche altri fattori che possono essere causa di effetto Forrester e che in genere sono sintomi di un'elevata oscillazione della tendenza **domanda/offerta**.

Eccessivo livello di scorte

Un tipico errore è quello di rifornire il magazzino in maniera eccessiva a livello di **scorte**, per evitare **rotture di stock**, ossia trovarsi senza scorte. In **Kanban** questo spesso si traduce con un dimensionamento eccessivo delle colonne di buffer,

dimensionamento che smorza l'effetto oscillante e che quindi impedisce di avere una misurazione in tempo reale delle fluttuazioni del mercato. Le notizie arrivano in ritardo, le contromisure che si intraprendono potrebbero non essere più in linea con quello che succede sul campo.

Inefficacia delle previsioni di vendita

L'inefficacia frequente o costante delle **previsioni di vendita** è un altro esempio. In agilità questo aspetto è stato abbondantemente affrontato: fare previsioni è molto difficile, se non impossibile, per cui si preferisce fare delle sperimentazioni in un ambito protetto e con un coefficiente di rischio ridotto al minimo (p.e.: iterazioni brevi, piccoli lotti di lavorazione, e così via).

Limitare gli sbalzi della capacità produttiva

La capacità produttiva **non costante** amplifica le oscillazioni che arrivano dal committente. Per questo si dovrebbe sempre cercare di stabilizzare il più possibile il contesto di lavoro: limitare il turn over delle persone, non variare la lunghezza degli sprint e comunque tenere sprint piccoli, livellare il flusso di alimentazione del backlog.

Anche **frequenti cambiamenti ai piani di produzione** alimentano l'effetto frusta. In un progetto software, il piano di produzione potrebbe essere qualcosa che ricade nel contesto Product Ownership o, più in alto, essere parte della vision. In Agile il cambiamento è benvenuto e per questo le metodologie come Scrum o Kanban riescono a reagire prontamente a eventuali variazioni.

È noto però che un cambiamento "forte" ha implicazioni non banali sul processo di produzione. Per esempio è probabile che la velocity cambi, o che il trend delle cose da fare, ossia il Burn Down di Scrum, subisca una radicale perturbazione.

Cause indirette

Oltre alle cause viste fino a questo punto, ci sono alcuni aspetti che forse è più corretto chiamare **effetti** ma che condizionano il sistema stesso dando luogo a un qualcosa che rientra nella teoria dei **system dynamics**, circuiti caratterizzati da ciclicità e feedback rientrante.

Comportamenti irrazionali

In determinati momenti, gli attori del sistema mettono in atto comportamenti irrazionali, frutto dell'emotività, o che possono apparirlo a posteri, come nel caso dei **decisori con razionalità limitata**. Si opera in un contesto a **razionalità limitata** durante il processo decisionale quando la razionalità un individuo è **limitata** dalle **informazioni** che **possiede**, dai limiti cognitivi della sua mente, e dall'**ammontare finito** di **tempo** che egli ha a disposizione per prendere una decisione.

Svariati modelli economici danno per scontato che le persone abbiano una razionalità media, e possano in quantità sufficientemente grandi essere approssimati come

agenti in accordo alle loro preferenze. Il concetto di razionalità limitata rivede questo assunto per tenere conto del fatto che **decisioni perfettamente razionali spesso non sono realizzabili nella pratica,** proprio a causa della quantità finita di risorse "computazionali" disponibili per prenderle [RAZ].

Mancanza di coordinamento

Spesso manca un coordinamento nelle decisioni dei singoli membri della catena di produzione. Anche in questo caso la mancanza di coordinamento è spesso legata ad una razionalità limitata dei decisori, i quali sono quindi portati ad assumere un **comportamento localmente opportunistico**: ogni decisore pensa prima di tutto a ottimizzare **il proprio limitato** ambito operativo.

In questo caso, poiché i decisori mancano delle capacità e delle risorse per arrivare alla soluzione ottimale, essi applicano invece la loro razionalità solo dopo aver enormemente semplificato le scelte disponibili. In pratica, il decisore cerca una **soluzione soddisfacente piuttosto che la migliore** in assoluto [RAZ].

Conclusione

L'effetto Forrester impatta sul processo di produzione e vendita di un bene, cosa che nel campo IT potrebbe essere la messa in produzione o il rilascio di un determinato componente o prodotto software.

La tipica conseguenza in una catena di produzione e vendita è che tutti gli attori nel mezzo della catena sono portati a un più o meno cosciente aumento delle scorte di sicurezza: si cerca di evitare la **rottura di stock**. La formica che accumula scorte per l'inverno, infatti, non ha problemi di obsolescenza del prodotto, cambio di requisiti — mangia sempre le stesse cose — o richieste particolari.

In un progetto software, il **magazzino** in genere è rappresentato da tutto il software che non è ancora stato rilasciato e che rimane in attesa di essere quindi verificato e provato sul campo. Per questo, in un progetto agile si cerca di minimizzare questa forma di magazzino, proprio per limitare l'accumulo di parti di prodotto non testate e non provate sul campo.

In un progetto software, gli accumuli di magazzino non sono né un obiettivo rincorso, né tantomeno una forma di protezione (irrazionale), ma casomai manifestazione di scarsa organizzazione o mancanza di strumenti: per esempio non si fa **continuous integration**.

Si è visto con l'esempio della produzione della birra che a un certo momento il sistema nel suo complesso ha iniziato a funzionare in modo non ottimale, cosa resa evidente da **ritardo nelle consegne** ad ogni livello della organizzazione. Questo effetto si può avere anche in una organizzazione che produce software. Si pensi per esempio a un sistema di gestione dei bug di produzione gestito tramite Kanban. In questo caso la variabilità può essere molto alta e, contrariamente a quanto possano dire gli analisti di mercato che si lanciano in rocambolesche previsioni, molto poco prevedibile.

Questo è un tipico caso in cui ciclo di lavorazione lungo, scarsa visione di insieme, ottimizzazioni localizzate, scarso coinvolgimento del cliente finale possono amplificare l'onda dell'effetto frusta, dando come risultato più evidente un ritardo nella lavorazione dei ticket.

Riferimenti

[5TH] Peter Senge, La quinta disciplina: L'arte e la pratica dell'apprendimento organizzativo, Sperling & Kupfer, 2006

[LBG] Che cosa è il beer game?
http://maaw.info/TheBeerGame.htm

[RAZ] La voce "Razionalità limitata" su Wikipedia
https://it.wikipedia.org/wiki/Razionalit%C3%A0_limitata

INDICE

Questo volume,
stampato nel mese di giugno 2020,
è la seconda versione del testo
Guida Galattica per Agilisti