Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

Modelli agili per lo sviluppo software: una panoramica su Scrum, Kanban e Scrumban

Relatore:
Chiar.mo Prof.
Paolo Ciancarini

Presentata da:
Luca Albonetti

Sessione III
Anno Accademico 2018-2019

Introduzione

Il processo di sviluppo software è una branca di studio dell'ingegneria informatica con lo scopo di indirizzare tutti gli individui coinvolti in un progetto a pratiche e metodi che migliorano le probabilità di buona riuscita del progetto stesso. Nel corso del tempo sono stati proposti svariati modelli di sviluppo software, con risultati variabili in rapporto alla grandezza del progetto, del team di sviluppo, del tempo e della disponibilità economica. Il compito del modello è quello di gestire al meglio il budget economico, il tempo a disposizione, la comunicazione e l'interazione tra gli sviluppatori, la piena comprensione dei requisiti e tutte le fasi che compongono il processo di sviluppo.

In tutti i campi vengono svolti studi per analizzare le differenze tra una qualsiasi azienda di settore e le aziende di maggior successo sul mercato, nel settore di riferimento, al fine di comprendere quali siano i processi aziendali che portano a risultati ottimali in maniera efficiente. Purtroppo, lo sviluppo di un software è un processo diverso dalla produzione: usando un'analogia con il mondo culinario potremmo dire che lo sviluppo software corrisponde a uno chef che inventa un piatto, mentre la produzione è più simile a un executive-chef che esegue una ricetta nel modo migliore possibile [3]. Questa semplice analogia evidenzia come lo sviluppo di un software sia legato alla creatività e all'esperienza, mentre la produzione sia un processo ripetitivo e meccanico; ovviamente, migliorare un processo produttivo è più semplice: dopo aver analizzato i vari passaggi e individuato i colli di bottiglia è pos-

sibile studiare un sistema che renda più efficiente la produzione lavorando sui punti critici; risulta più complesso, invece, studiare dei sistemi sempre efficienti per lo sviluppo software, poiché l'efficacia del metodo è legata non solo alla struttura aziendale, ma anche alla cultura aziendale e al metodo operativo adottato dal team di sviluppo. Per questo motivo esistono molti metodi legati allo sviluppo software che hanno obiettivi simili, ma impostazioni completamente differenti.

In linea generale il processo di sviluppo software è caratterizzato dalle seguenti fasi e strumenti:

- Cattura, specifica, analisi e gestione dei requisiti
- Progettazione dell'architettura e dei moduli
- Codifica e debugging
- Testing
- Deployment

Ogni modello di sviluppo software è caratterizzato dalle specifiche che fornisce per ognuna delle fasi illustrate precedentemente; poiché non esiste un modello che sia più efficace degli altri in ogni situazione (no silver bullet), scegliere la metodologia e il metodo più adeguati al sistema in esame diventa, quindi, un fattore decisivo, per la buona riuscita del prodotto finale.

In molti casi i metodi di sviluppo software traggono la loro ispirazione da metodi che sono già stati applicati in ambito produttivo; trasferendo i principi dalla produzione allo sviluppo è possibile applicare gli stessi metodi, prestando attenzione ad adeguare le pratiche, infatti i principi sono universali, mentre le pratiche corrispondono all'applicazione dei principi nel dominio [3].

Le principali metodologie di sviluppo software possono essere catalogate nel modo seguente:

- metodologie pesanti (es. modello a cascata)
- metodologie iterative (es. modello a spirale)
- metodologie agili (es. Scrum, XP, Kanban, ...)

Negli ultimi anni le metodologie agili hanno conosciuto una grande diffusione, lo studio "VersionOne State of Agile Development Survey 2013" riporta che nel 2013 molti team di sviluppo hanno tratto giovamento dall'uso di una metodologia agile [2]:

- l'88% di chi ha risposto dice che la sua azienda usa una metodologia agile
- il 92% di chi ha risposto ha riscontrato un miglioramento di anno in anno in tutte le aree individuate dal questionario, in particolare le categorie in cui ci sono stati miglioramenti sono: abilità nel gestire i cambiamenti (92%), aumento di produttività (87%), maggior visibilità del progetto (86%), maggiore motivazione per il team (86%) e una migliore qualità del software (82%)

I motivi di questo successo sono individuabili nei principi stessi della metodologia agile:

- La nostra massima priorità è soddisfare il cliente rilasciando software di valore, fin da subito e in maniera continua.
- Accogliamo i cambiamenti nei requisiti, anche a stadi avanzati dello sviluppo. I processi agili sfruttano il cambiamento a favore del vantaggio competitivo del cliente.
- Consegnamo frequentemente software funzionante, con cadenza variabile da un paio di settimane a un paio di mesi, preferendo i periodi brevi.

- Committenti e sviluppatori devono lavorare insieme quotidianamente per tutta la durata del progetto.
- Fondiamo i progetti su individui motivati. Diamo loro l'ambiente e il supporto di cui hanno bisogno e confidiamo nella loro capacità di portare il lavoro a termine.
- Una conversazione faccia a faccia è il modo più efficiente e più efficace per comunicare con il team ed all'interno del team.
- Il software funzionante è il principale metro di misura di progresso.
- I processi agili promuovono uno sviluppo sostenibile. Gli sponsor, gli sviluppatori e gli utenti dovrebbero essere in grado di mantenere indefinitamente un ritmo costante.
- La continua attenzione all'eccellenza tecnica e alla buona progettazione esaltano l'agilità.
- La semplicità - l'arte di massimizzare la quantità di lavoro non svolto - è essenziale.
- Le architetture, i requisiti e la progettazione migliori emergono da team che si auto-organizzano.
- A intervalli regolari il team riflette su come diventare più efficace, dopodiché regola e adatta il proprio comportamento di conseguenza.

[6]

Questi fondamenti sono antitetici rispetto a quelli delle metodologie pesanti, che invece propendono per una impostazione più classica (dal punto di vista del management aziendale).

Una metodologia agile è costituita da un insieme di pratiche che producono buoni risultati se applicate contemporaneamente [2]; il focus di ogni metodologia agile sono la qualità e la comunicazione, sia con il cliente che

tra i membri del team di sviluppo; il risultato di questo approccio è quello di produrre un servizio di cui, molto spesso, il cliente è più soddisfatto rispetto a uno stesso servizio generato seguendo una metodologia pesante [3]. La necessità di dover adattare il software alle richieste del cliente e alla continua evoluzione degli strumenti e delle tecniche di sviluppo software costituisce un ulteriore incentivo per le aziende ad adottare un metodo agile.

> "Nell'economia dei servizi, qualità non significa rispettare un contratto; significa adattarsi alle aspettative, in continuo cambiamento, di molti clienti." [3]

È importante notare come tra i principi agili non vi sia alcun riferimento alla organizzazione interna o alla struttura gerarchica, mentre l'attenzione è completamente rivolta alla comunicazione e collaborazione; infatti lo sviluppo software non è pensato per riprodurre risultati ripetibili, ma per fornire soluzioni adeguate e costruite su misura per il cliente [3].

Questo obiettivo viene raggiunto tramite le iterazioni. Un'iterazione è un incremento utile del software, pensato, programmato, testato, integrato e rilasciato in un breve periodo di tempo. Il software verrà migliorato con il susseguirsi delle iterazioni, ma è funzionante già dal primo rilascio. Le iterazioni consentono di avere un feedback continuo con il cliente, mentre nei metodi pesanti non si ha feedback fino al rilascio finale [3].

Le iterazioni sono fondamentali per la comunicazione non solo con il cliente, ma anche tra i membri del team di sviluppo; infatti, una volta completata una feature è necessario integrarla nel sistema al fine di creare un prodotto uniforme e funzionante [3]. Inoltre le iterazioni permettono di gestire meglio i cambiamenti, avere tempi noti di consegna parziale del software permette di intervallare la parte implementativa con la pianificazione fornendo dei momenti in cui è possibile valutare modifiche rispetto ai piani stabiliti precedentemente. Gestire il cambiamento significa anche documentarlo: un tipico

problema delle metodologie pesanti è quello di avere una documentazione ben dettagliata ma esageratamente grande, difficile da gestire e modificare, invece le metodologie agili propendono per una documentazione più concisa che contenga tutte le informazioni necessarie per far funzionare il progetto, anche se è possibile ampliare i confini della documentazione in base alle necessità del team di sviluppo [2].

L'idea è quella di implementare un insieme coerente di feature a ogni iterazione. Una feature è un'attività che aggiunge valore al software dal punto di vista del cliente, ma è sufficientemente piccola da permettere al team di sviluppo di stimarne il tempo di implementazione in modo accurato; quando una feature è troppo complessa per poter essere rilasciata in una sola iterazione deve essere spezzetata in feature più piccole [3].

Le continue modifiche apportate al software per introdurre nuove feature rischiano, però, di minare la stabilità del prodotto. Per questo motivo i team agili fanno ampio uso dei test. Questi ultimi sono utili per comunicare in modo inequivocabile quali funzionalità ci si aspetta, inoltre garantiscono che tutte le parti del software funzionino correttamente e che il sistema sia effettivamente in grado di rispettare le aspettative.

La pratica di eseguire tutti i test prima di ogni rilascio è nota con il nome di integrazione continua (continuous integration) ed è diffusa in tutti i metodi agili. Oltre a garantire la qualità del software, i test costituiscono anche una documentazione riguardo al design dello stesso. Infatti, la documentazione dei metodi pesanti risulta spesso inutile perché non rispecchia il vero design del software: a volte la documentazione non viene aggiornata in fase di implementazione con il risultato che diventa subito obsoleta. Invece, sostituendo parte della documentazione con i test non si riscontra questo problema poiché i test, per essere utili, devono sempre essere aggiornati.

Nei prossimi capitoli analizzeremo nel dettaglio tre metodi agili: Scrum, Kanban e Scrumban. Quest'ultimo è nato dalla fusione dei primi due metodi. L'aspetto interessante che nasce dal confronto tra Scrum e Kanban è che sono entrambi metodi agili, ma con approcci differenti: Scrum definisce delle procedure di sviluppo, mentre Kanban fornisce degli strumenti per migliorare il processo di sviluppo.

Indice

Elenco delle figure

Capitolo 1

Scrum

Scrum è nato nel 1995 ad opera di Jeff Sutherland e Ken Schwaber ed è il metodo agile che ha riscosso più successo tra le aziende negli ultimi anni; le caratteristiche che ne hanno decretato il successo sono essenzialmente due: la distinzione dei ruoli e la scalabilità.

Infatti, nelle specifiche di Scrum sono definiti dei ruoli precisi per i partecipanti, questo rende più semplice per un'azienda adattare la sua struttura gerarchica al metodo Scrum, mentre modelli agili alternativi come l'Extreme Programming (XP) sono più liberi da vincoli e quindi più difficili da integrare con una tipica struttura gerarchica aziendale.

Per quanto riguarda la scalabilità, è possibile affermare che, anche se Scrum è definito per un gruppo di lavoro ristretto (una decina di persone circa), è possibile applicare Scrum anche a gruppi di lavoro più ampi: lo Scrum di Scrum consiste nel suddividere tutti i partecipanti in Team Scrum, lo Scrum Master di ogni team fa parte di un Team Scrum di livello superiore, e così via fino a raggiungere il vertice della scala gerarchica.

1.1 Il metodo Scrum

Scrum è un metodo iterativo e incrementale, basato su controllo empirico del processo; è incrementale perché il prodotto finale è il risultato di continui rilasci, ciascuno dei quali aggiunge valore a quello precedente; è iterativo perché ogni nuovo rilascio viene adattato in base alle contingenze del momento [2].

I suoi fondamenti sono: un processo di sviluppo con fasi ben caratterizzate, una definizione precisa dei ruoli e una serie di rituali.

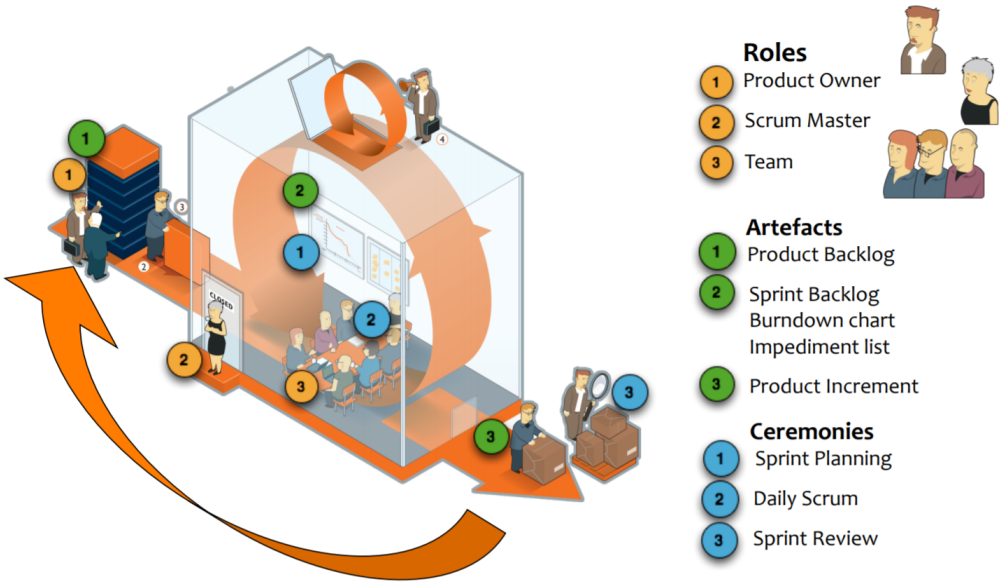

Figura 1.1: Il processo Scrum [16]

1.1.1 Il processo

Il processo Scrum è caratterizzato da una fase iniziale di raccolta delle feature per il progetto richieste dal cliente che vengono riportate con delle User Stories: una User Story è una breve descrizione di come l'utente intende

usare la feature richiesta. La scelta di quali User Stories accettare viene svolta esclusivamente dal Product Owner, che le inserisce nel Product Backlog. All'interno del Product Backlog le User Stories possono essere ulteriormente spezzetate dal team di sviluppo per creare dei requisiti più piccoli, la ratio è quella di rendere lo sforzo richiesto dall'implementazione di ogni requisito adattabile ai tempi delle iterazioni.

Successivamente il Product Owner sceglie tra i requisiti individuati nel Product Backlog quelli che si vogliono sviluppare nella successiva iterazione della fase di sviluppo, nel gergo Scrum i requisiti relativi a una singola iterazione vengono chiamati Sprint Backlog. Ogni iterazione è definita da uno Sprint: un periodo di tempo, solitamente compreso tra due e quattro settimane, durante il quale si sviluppano i requisiti che sono stati inseriti nello Sprint Backlog; ogni Sprint comprende l'implementazione di nuove feature e i relativi test. Lo Sprint è la fase più delicata per lo sviluppo, quindi non sono ammesse modifiche allo Sprint Backlog e al Team Scrum fino al termine dello stesso.

Al termine dello Sprint segue una fase di revisione, detta Sprint Review, in cui si analizzano i risultati ottenuti e cosa è andato storto durante la fase di sviluppo. L'obiettivo è quello di migliorare il processo di sviluppo per le iterazioni future e accertarsi di aver soddisfatto le aspettative del cliente. Inoltre, comprendere quanto valore è stato aggiunto al prodotto ad ogni iterazione rende i membri dello Scrum Team più motivati e interessati al progetto. In questa fase il team verifica la qualità della release attraverso i test, in questo modo si accerta la continuità nel funzionamento generale del software e la correttezza di quanto prodotto durante lo Sprint.

Una volta completata la fase di Sprint Review si ritorna al Product Backlog e si scelgono i requisiti da implementare nello Sprint successivo.

Potrebbe sembrare che in Scrum ci sia poca pianificazione, invece questa è solo un'impressione dovuta al fatto che la pianificazione non viene svolta in modo completo prima di cominciare a implementare i requisiti, ma viene diluita nello Sprint: un tipico Team Scrum spende 8 ore per pianificare l'iterazione, poi adatta la pianificazione durante i Daily Meetings (15 minuti per 30 giorni); se si considera un team di 5 persone, si scopre che la pianificazione comporta uno sforzo di 40 persone/ora per iterazione e altre 40 persone/ore di pianificazione spalmata sui successivi 30 giorni, in conclusione la quantità di tempo dedicata alla pianificazione risulta essere ben maggiore rispetto ad altri metodi, sempre considerando 30 giorni di sviluppo [2].

1.1.2 I ruoli

Il punto focale di questo metodo è il Team Scrum, che è composto da:

- Product Owner
- Development Team
- Scrum Master

Il Product Owner è la figura che rappresenta gli stakeholder (quindi il cliente) all'interno del contesto Scrum; si occupa della gestione delle relazioni esterne ed è colui che stila il Product Backlog, una raccolta di tutte le User Stories del progetto; inoltre, decide quali requisiti inserire in ogni iterazione, dopo aver verificato con il Development Team che le feature scelte siano effettivamente realizzabili nei tempi previsti. Il suo ruolo è decisivo per la buona riuscita del progetto poiché risponde ai dubbi del Development Team in vece del cliente, la sua comprensione del progetto è, quindi, essenziale per fare le scelte corrette durante lo sviluppo. Il Product Owner ha anche la possibilità di interrompere uno Sprint se si accorge che è stato pianificato malamente o se sopraggiungono dei cambiamenti che richiedono un immediato intervento; l'interruzione di uno Sprint è un gesto drastico, un evento eccezionale che non andrebbe ripetuto troppo spesso perché minerebbe la fiducia nel progetto sia

per gli sviluppatori sia per gli stakeholder.

Il Development Team è costituito da un gruppo di sviluppatori, ognuno con competenze specifiche diverse; solitamente il team è composto da un numero di membri variabili tra 3 e 9.

Lo Scrum Master è la figura che più caratterizza il metodo Scrum: il suo compito è quello di agevolare il lavoro del team occupandosi di rimuovere tutti gli ostacoli per il raggiungimento dell'obiettivo dello Sprint; inoltre, costituisce il canale di comunicazione tra Product Owner e Development Team ed è responsabile per la corretta esecuzione delle pratiche Scrum. Lo Scrum Master ha un ruolo particolare perché non agisce come superiore (nella scala gerarchica) ma come leader, allenatore e shepard per il team attraverso tutta l'evoluzione del progetto [2]. Infatti, il compito dello Scrum Master non è quello di creare un piano di sviluppo, ma di studiarne uno con il Development team e agevolarne l'attuabilità; il punto è che nessuno è univocamente responsabile per eventuali ritardi, ma tutto lo Scrum Team si fa carico di questo fardello [2].

1.1.3 I rituali

Una parte importante del metodo Scrum è quella costituita dai suoi rituali:

- Sprint Planning Meeting
- Daily Scrum (noto anche come Stand-up Meeting)
- Sprint Review
- Sprint Retrospective

La fase di Sprint Planning precede ogni iterazione ed è funzionale alla scelta effettuata dal Development Team di quali requisiti implementare durante lo Sprint. È anche possibile che alcuni requisiti nel Product Backlog

vengano modificati.

Il Daily Scrum consiste in un meeting di breve durata (solitamente 15 minuti) in cui ogni membro del Development Team specifica cosa vuole sviluppare in giornata e illustra ai compagni eventuali problemi riscontrati durante lo sviluppo. Inoltre, parlare tutti i giorni dei compiti svolti da ciascun membro del team rende più facile per il singolo sviluppatore capire cosa sta facendo il resto del team per il progetto e a quale punto dello sviluppo si è giunti. In questa fase ogni sviluppatore sceglie di quale requisiti farsi carico per la giornata, in questo modo tutti i membri del team sanno su cosa lavoreranno i loro colleghi. Il Daily Scrum non serve tanto a risolvere i problemi quanto a individuarli e capire quale sia il momento e il modo più adatto per affrontarli; la soluzione effettiva al problema viene discussa dagli sviluppatori interessati in un meeting a cui il resto del team non partecipa per evitare di bloccare tutto il team di sviluppo su un unico requisito [2].

Lo Sprint Review e lo Sprint Retrospective avvengono al termine di ogni iterazione; durante lo Sprint Review si valuta la demo risultante dallo sviluppo, mentre la Sprint Retrospective è il momento nel quale si analizza ciò che è stato sviluppato con successo e quali problemi sono stati riscontrati per ciò che non è stato sviluppato. La fase di Sprint Review risulta utile a diversi scopi: comunicazione con gli stakeholder, comunicazione tra i membri del team, che possono vedere i risultati del proprio lavoro, infine, l'integrazione delle diverse feature sviluppate all'interno di una demo permette di individuare eventuali problemi prima di aggiungere ulteriori elementi al progetto [2].

Lo scopo di questi rituali non è tanto nel valutare i progressi fatti per valutare gli sviluppatori, ma quello di instaurare un legame tra gli stessi e imparare dagli errori commessi per migliorare il processo di sviluppo.

I rituali sono necessari ma non sufficienti per usare il metodo Scrum; se vengono usati nel modo sbagliato possono risultare comunque efficaci ma non in grado di sfruttare a pieno le potenzialità legate a una buona applicazione dei principi agili. Un esempio potrebbero essere quello dei Daily Scrum usati non come momento di collaborazione e condivisione di informazioni, ma come occasione di scontro e delimitazione delle responsabilità, sicuramente questo tipo di interazione non fortifica i legami tra gli sviluppatori. In questi casi si può dire che l'organizzazione si avvantaggia delle pratiche Scrum, ma non usa il metodo Scrum [2]; applicare correttamente la filosofia agile costituisce la parte più difficile, soprattutto nelle fasi iniziali, ma col tempo contribuisce a consolidare i metodi e le pratiche e li rende efficaci al massimo delle loro potenzialità.

1.2 Strumenti per implementare Scrum

Gli artefatti prodotti dalle varie fasi di sviluppo sono:

- Product Backlog
- Sprint Burndown Chart
- Sprint Backlog
- Scrum Task Board

Il Product Backlog è costituito dall'insieme di tutte le User Stories che riguardano il prodotto in sviluppo; quali requisiti possano entrare a far parte del backlog e con quale priorità è una scelta che viene effettuata esclusivamente dal Product Owner poiché è il membro dello Scrum Team più vicino al cliente e alle sue esigenze.

Lo Sprint Backlog è formato da alcune User Stories selezionate dal Product Backlog, all'inizio di ogni Sprint il Product Owner decide quali requisiti hanno priorità maggiore e devono essere, quindi, terminati entro lo Sprint;

nonostante l'ultima parola spetti al Product Owner, egli fa le sue valutazioni anche sulla base di quale sia il carico di lavoro che il Development Team e lo Scrum Master ritengono adeguato per lo Sprint. In questo senso il Development Team è auto-organizzato: capisce quale sforzo è in grado di sostenere per la durata dello Sprint e come suddividere i compiti in fase di sviluppo, senza influenze esterne.

La Sprint Burndown Chart e lo Scrum Task Board sono degli strumenti fondamentali per il Development Team poiché facilitano la comunicazione tra gli sviluppatori: la Scrum Task Board risulta utile per capire quale membro del team stia lavorando a una feature e a quale fase dello sviluppo sia giunto, invece la Sprint Burndown Chart illustra qual'é lo stato di avanzamento dello Sprint e può essere utile per capire quali task sono rimasti bloccati e hanno bisogno di uno sforzo maggiore per essere terminati rispetto alle promesse iniziali. Entrambi questi due strumenti uniti ai Daily Meeting completano la comunicazione tra i membri del team perché questi elementi permettono una comunicazione visiva, mentre il Daily Meeting aggiunge la comunicazione verbale.

La Sprint Burndown Chart è un grafico che illustra l'andamento dello Sprint: sull'asse delle ascisse vengono indicati i giorni dedicati allo Sprint, sull'asse delle ordinate viene misurato il numero di requisiti che non sono ancora stati completati; la retta che viene disegnata su questo grafico parte dal numero di requisiti dello Sprint sull'asse delle ordinate per procedere in diagonale verso il basso. Risulta quindi evidente per chiunque, con un semplice colpo d'occhio capire, se lo sviluppo prosegue come nelle previsioni o se gli sviluppatori sono bloccati (in questo caso troveremo una linea costante, in alcuni casi addirittura crescente) [2].

La Sprint Burndown Chart può essere costruita automaticamente tramite l'uso di software, ma molti team preferiscono mantenerla in versione fisica

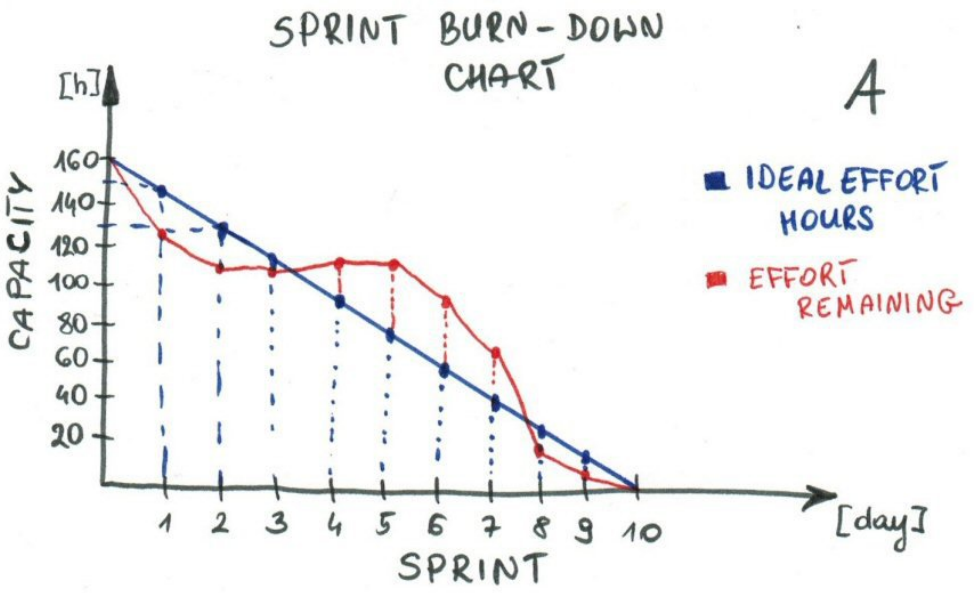

Figura 1.2: Sprint Burndown Chart [2]

vicino alla Task Board per lasciare agli sviluppatori maggiore soddisfazione nel vedere che il risultato del loro lavoro aiuta il team a raggiungere l'obiettivo; inoltre un elemento fisico presente nella stessa stanza dove lavora il Development Team permette a tutti di mantenere sempre sotto controllo l'avanzamento dello sviluppo.

La Scrum Task Board è una lavagna divisa in colonne: To-Do, In-Progress, Done. Inizialmente vengono poste nella colonna To-Do tutte le User Stories che dovranno essere implementate durante lo Sprint, vicino a ogni User Story vengono collocati tutti i requisiti, sempre tra quelli scelti per lo Sprint, che la riguardano. Ogni sviluppatore decide durante il Daily Scrum quali requisiti sviluppare nel corso della giornata, quindi appena comincia a lavorare su un requisito lo sposta dalla colonna To-Do a In-Progress e scrive il suo nome sul task scelto, infine, quando ha terminato l'implementazione lo sposta in Done. Solitamente uno sviluppatore lavora su un solo requisito alla volta per mantenere la sua concentrazione sul task più alta possibile.

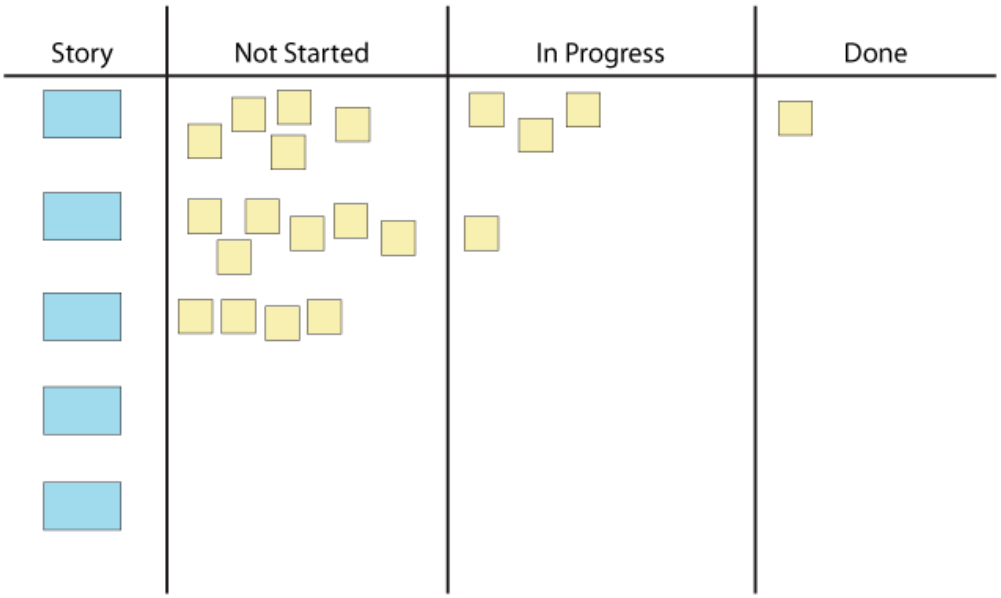

Figura 1.3: Scrum Task Board [2]

A volte capita che gli sviluppatori individuino dei requisiti aggiuntivi che sono necessari per completare una User Story, ma che non erano stati considerati inizialmente; in questi casi è sufficiente che lo sviluppatore informi i suoi colleghi al Daily Meeting in modo che anche gli altri possano valutare l'inserimento del nuovo task e capire se entra in conflitto con quanto è stato fatto o si era programmato di fare. Al termine dello Sprint, tutti i requisiti che si trovano nella colonna Done possono essere eliminati, se una User Story è stata completata, poiché non ha più task aperti, può essere eliminata; invece, tutte le User Stories e i requisiti che si trovano in To-Do e In-Progress vengono reinseriti nel Product Backlog per continuare il loro sviluppo in un nuovo Sprint [2].

Oltre agli strumenti che abbiamo già analizzato ne esistono altri che non farebbero parte del metodo Scrum per come era stato pensato inizialmente, ma che con il tempo sono entrati a far parte della collezione degli strumenti

di Scrum: sono i cosiddetti Generally Accepted Scrum Practices (GASPs). Questi strumenti possono comprendere piccole variazioni a come era stato pensato Scrum, per esempio il Daily Scrum viene svolto in piedi da alcuni team e viene chiamato Stand-up Meeting, oppure possono consistere nell'inserimento di metodi aggiuntivi, un esempio per tutti è un metodo per la stima dello sforzo di un requisito noto con il nome di Planning Poker. Quest'ultimo consiste nello stabilire una stima dello sforzo per un requisito tramite un metodo di votazione tra gli sviluppatori: ogni votante ha un mazzo di carte, ognuna con un valore diverso (i valori sono arbitrari, il loro significato è quello di indicare dei livelli di complessità), e sceglie una carta per il requisito in esame; capita spesso che i valori scelti dagli sviluppatori siano molto discordanti tra loro poiché ognuno di essi fa valutazioni con criteri propri, un confronto tra i votanti è utile per chiarire la corretta comprensione del requisito e quindi stabilire una valutazione soddisfacente per tutti gli sviluppatori.

> "A intervalli regolari, il team riflette su come diventare più efficiente, poi modifica e aggiusta il suo metodo operativo di conseguenza" [6]

Adattare le procedure e gli strumenti di un metodo agile alle esigenze del proprio team è una pratica che viene incoraggiata nel mondo Agile; infatti, quando al termine dello Sprint si fa una retrospettiva di cosa ha funzionato e di cosa è andato storto è utile pensare a come rendere il team più produttivo, per raggiungere questo fine la metodologia agile non fornisce linee guida e lascia, quindi, spazio alla fantasia e alla capacità dei membri del team nel trovare una soluzione efficace.

1.3 Le metriche di Scrum

Le fasi finali di ogni Sprint permettono allo Scrum Team di intraprendere un percorso di continuo miglioramento del processo di sviluppo. Lo scopo

è quello di migliorare la qualità del software, cioè di soddisfare i requisiti richiesti dal cliente.

Per comprendere quale sia il livello di qualità raggiunto e quali siano i margini di miglioramento è necessario raccogliere dei dati riguardo agli obiettivi raggiunti e a quelli mancati, andando così a individuare le aree più critiche del processo di sviluppo.

La Sprint Burndown Chart è uno strumento che permette di misurare lo sforzo che è stato necessario per portare a termine le User Stories e le attività durante lo Sprint. La forma di questo grafico è già stata spiegata precedentemente, in questa sezione ne evidenziamo solamente la capacità di illustrare il divario tra le aspettative e l'effettiva capacità del team di completare i compiti previsti.
Il fatto che un team non sia stato in grado di rispettare le previsioni può essere dovuto semplicemente al fatto che alcune attività sono state sottostimate, oppure al fatto che ne sono state introdotte troppe nel corso dello Sprint; quest'ultima situazione è dovuta al fatto che il Product Owner ha inserito troppo carico di lavoro sulle spalle del team e lo Scrum Master non è stato in grado di salvaguardare il team da un eccessivo stress, migliorando la comunicazione tra queste due figure è possibile evitare che situazioni simili si ripetano.

La Release Burndown Chart è uno strumento molto utile per misurare la velocità del team di rilasciare le attività nel Product Backlog e tenere traccia di quanto lavoro rimane per completare il prodotto.
Questo grafico rappresenta sull'asse delle ascisse il numero di Sprint effettuati e sull'asse delle ordinate lo sforzo richiesto complessivamente dalle attività nel Product Backlog. La misura dello sforzo può essere misurato nel modo che il team preferisce, per esempio: punteggio assegnato alla User Story, dimensione dell'attività, ore di lavoro.

Affinché la Release Burndown Chart sia utile diventa necessario:

- specificare il numero di release da osservare e quali sono le date di inizio e di fine per ognuna di esse
- definire le attività nel Product Backlog e assegnarle a ogni Sprint
- all'inizio di ogni Sprint si fa una stima del carico di lavoro per la release
- al termine dello Sprint si rimuovono le attività completate

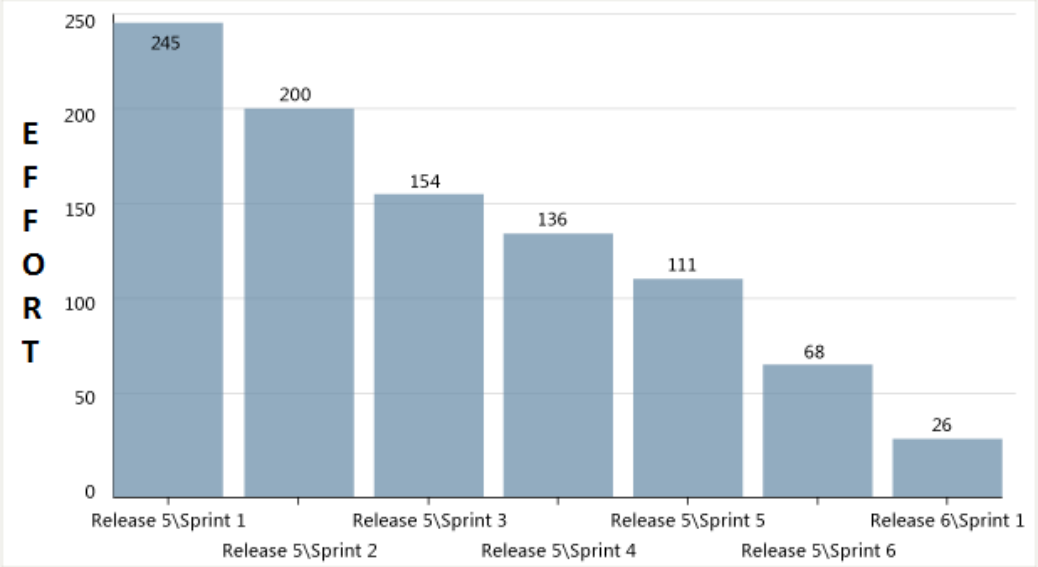

Figura 1.4: Release Burndown Chart [15]

Questo strumento è utile per tracciare l'andamento del progetto e avere uno storico di quante feature siano state inserite tra uno Sprint e l'altro nel corso del tempo e quante sono state completate.

Un altro strumento molto utile per i team Scrum è il Velocity Report. Questo grafico è simile alla Release Burndown Chart, ma invece di misurare il numero complessivo di attività presenti nel Product Backlog registra lo sforzo dei compiti portati a termine ad ogni rilascio.
Il Velocity Report permette di stimare una media dello sforzo che il team è

in grado di sostenere per ogni Sprint, a patto che i componenti del team e la durata degli Sprint rimangano invariati. Inoltre, questo grafico rende subito evidente l'andamento delle performance del team, quando si verificano delle inflessioni è chiaro che il processo di sviluppo richiede una revisione.

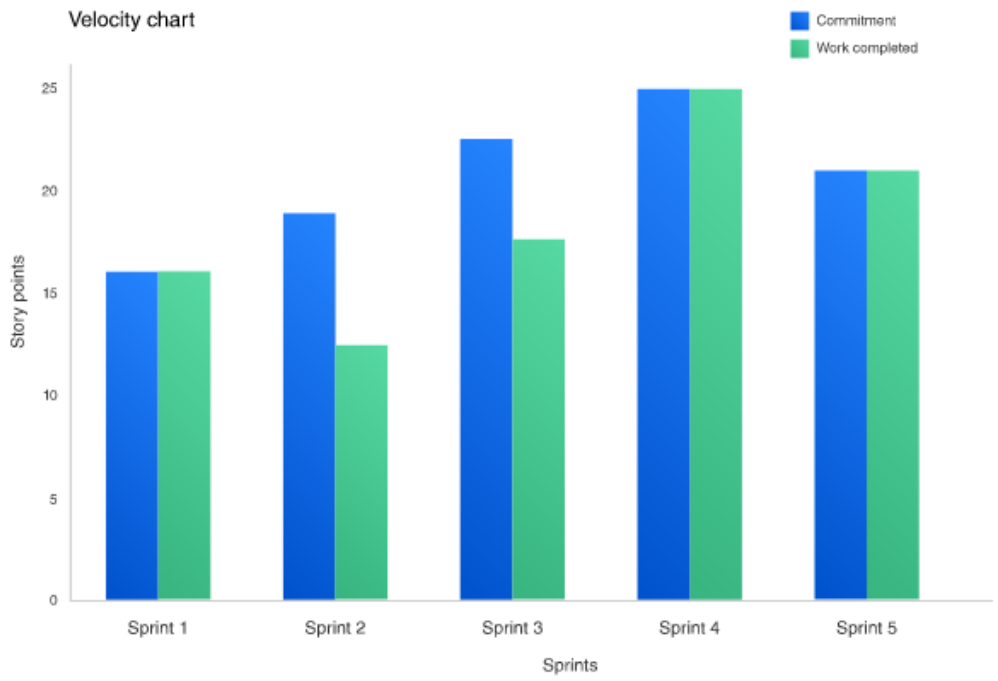

Figura 1.5: Velocity Report [17]

Altre metriche più semplici, ma comunque interessanti comprendono:

- User Stories completate vs. User Stories assegnate allo Sprint: si misura l'abilità del team di comprendere e prevedere le sue capacità confrontando il numero di User Stories assegnate allo Sprint con quelle effettivamente completate.
- Technical Debt Management: comprende i problemi noti e i bug inclusi nel rilascio al termine dello Sprint; solitamente misura il numero di bug, ma può anche includere altre informazioni come la documentazione e

i mezzi usati per il rilascio; le User Stories che vengono inserite in questa categoria possono essere considerate completate rispetto alle aspettative del cliente, ma che sono state implementate in modo non ottimale per riuscire a rispettare i tempi di consegna.

- Team Velocity: corrisponde alla media di quanto sforzo il team riesce a compiere in un singolo Sprint (vengono considerate solo le attività completate e prive di difetti e/o bug); viene calcolato confrontando il risultato di ogni Sprint con quello precedente, l'obiettivo è di mantenere una capacità costante con margini di +/-10%.

- Qualità del prodotto consegnato al cliente: si valuta se il software rispetta le aspettative e i bisogni del cliente, si cerca di capire quale sia il contributo di ogni Sprint nell'aggiungere valore al prodotto; al termine di ogni Sprint si fornisce una demo al cliente e in base al suo feedback è possibile stimare quale sia il suo grado di soddisfazione.

- Entusiasmo del team: mantenere monitorato il grado di entusiasmo del team per il progetto è utile perché senza questa componente non esiste processo o metodologia in grado di salvare la situazione; questa metrica può essere ottenuta osservando i risultati degli Sprint nel corso del tempo o chiedendo direttamente agli sviluppatori quanto sono soddisfatti del progetto e si sentano motivati; un altro parametro da tenere in considerazione è il turnover del team, poiché un valore alto può denotare una scarsa motivazione tra i suoi membri.

- Retrospective Process Improvement: l'abilità di uno Scrum Team di rivedere il suo processo di sviluppo per renderlo più efficace nello Sprint successivo; può essere misurato contando quante attività sono state sottoposte a revisione, quante di queste il team si è impegnato a risolvere e quante attività sono state risolte entro il termine dello Sprint.

- Comunicazione: è possibile valutare quanto il Development Team, il Product Owner, lo Scrum Master e il cliente riescono a comunicare tra

di loro in modo aperto e onesto; l'unico modo per stimare la qualità della comunicazione è quella di osservare e ascoltare.

- In quale misura il team ha aderito alle pratiche e al metodo Scrum: anche se Scrum non definisce delle pratiche ingegneristiche universali, molte aziende ne stabiliscono delle proprie; è possibile verificare quanto spesso, nel corso di uno Sprint, il team non rispetti le regole prestabilite per misurare il suo grado di aderenza alla metodologia.

- Sprint goal success rates: misura il grado di successo raggiunto dal team in ogni Sprint, per raggiungere un tasso soddisfacente non è necessario portare a termine tutte le attività nello Sprint Backlog, ma una quantità prestabilita di esse e che siano considerabili completate a tutti gli effetti (test, documentazione e integrazione con le altre feature).

- La comprensione dell'obiettivo dello Sprint da parte del team: è una valutazione soggettiva di quanto il team e il cliente abbiano compreso l'obiettivo dello Sprint; solitamente l'obiettivo corrisponde al valore aggiunto che il cliente si aspetta e che è stato definito nei criteri di accettazione per la User Story. Questa valutazione può essere effettuata solo mantenendo un contatto quotidiano con il team e con il cliente.

Capitolo 2

Kanban

Kanban è un metodo adattato dal mondo della produzione a quello dello sviluppo software da David Anderson. Kanban ha un obiettivo diverso rispetto ad altre metodologie agili come Scrum: mentre quest'ultima si concentra sul management, Kanban si focalizza su come migliorare il processo di sviluppo.

> "Kanban non è una metodologia di sviluppo software o un approccio al management. Richiede che esista già un processo, così che Kanban possa essere applicato per migliorare in modo incrementale il processo sottostante." David Anderson

Kanban è un metodo agile che mette in pratica anche i principi Lean. Kanban si distingue da Scrum in quanto non è una metodologia, ma un modo di pensare: ha dei valori e dei principi ma non ha pratiche; i suoi principi sono ispirati da sette accorgimenti derivati da Lean per ridurre gli sprechi di risorse e tempo nello sviluppo software [2]:

- eliminare gli sprechi
- amplify learning (diffondere le conoscenze apprese)
- decidere all'ultimo momento utile
- rilasci frequenti

- fornire potere decisionale e fiducia al team di sviluppo
- costruire software coerente
- avere la visione d'insieme

Per ognuno di questi valori esiste un insieme di strumenti applicabili nel mondo reale e che verranno analizzati in seguito.

Poiché Kanban necessita di un sistema già in essere e fornisce strumenti per migliorare il processo del team di sviluppo, un team che faccia uso di Kanban deve avere una idea chiara di quali siano i passaggi che vengono svolti per costruire il software e come ogni operazione vada a influire sugli altri reparti dell'azienda, dove e quando avvengono sprechi, e come migliorare il processo rimuovendo le cause delle inefficienze. Kanban è un esempio di come le idee agili possano essere applicate per migliorare un processo rendendolo più lineare e semplice per il team [2].

I principi fondamentali di Kanban sono:

- cominciare dal processo già in essere
- tutti i membri del team concordano per intraprendere un percorso di miglioramento incrementale
- almeno nella fase iniziale è necessario rispettare i ruoli e le responsabilità già in essere

In pratica, l'obiettivo di un team che adotta il metodo Kanban è quello di intraprendere un percorso di miglioramento incrementale con lo scopo di individuare i problemi ricorrenti, trovare i fattori comuni e cercare una soluzione per risolverli. La fase di individuazione del problema è quella più critica, perché un team può essere abituato ad affrontare problemi ricorrenti che nel lungo periodo non vengono più visti come errori, ma come effetti

collaterali inevitabili.

Kanban comincia proprio da questa fase: capire come si lavora per suddividere l'intero processo in fasi ricorrenti, ma modificabili. In Kanban le fasi del processo di sviluppo o le regole vengono chiamate Policies. Come primo passo un team deve riconoscere le proprie abitudini, identificare le fasi che ricorrono ogni volta che costruiscono un software e riportare il tutto nella documentazione. Ogni team ha un sistema con cui costruisce il software, che può anche non essere rigoroso e può cambiare frequentemente, ma esiste comunque, anche solo nella mente degli sviluppatori che lo usano. I team che usano delle metodologie, come Scrum, hanno un sistema codificato e ben compreso da tutti, ma molti team seguono più semplicemente un sistema che esiste come conoscenza "tribale". Una volta che il team ha definito le varie fasi, Kanban migliora il processo in piccoli passi, per questo motivo si parla di miglioramenti incrementali. Per poter valutare il risultato ottenuto dalle modifiche apportate è necessario misurare i rendimenti nelle fasi prima e dopo ogni modifica per poter stabilire se gli effetti ottenuti sono quelli cercati e Kanban fornisce molti strumenti per effettuare queste valutazioni.

Quindi, anche se Kanban non è un metodo gestionale, ha una forte connessione con il metodo gestionale usato dal team poiché ogni cambiamento al processo di sviluppo ha delle ripercussioni sulla gestione complessiva dell'organizzazione.

Esiste una differenza tra Kanban e i tradizionali metodi di miglioramento del processo produttivo. Nei modelli tradizionali, le decisioni vengono prese dal dirigente e comunicate ai suoi sottoposti fino a raggiungere la base della scala gerarchica. Invece, nel metodo Kanban i cambiamenti vengono scelti ed effettuati dal team di sviluppo, quindi il team trova da solo i problemi nel suo processo di sviluppo e propone delle modifiche, infine misura i risultati ottenuti; conseguentemente tutto il team è responsabile per i miglioramenti

o peggioramenti dei propri risultati.

2.1 Il metodo Kanban

2.1.1 Il processo

Quando un team vuole adottare il metodo Kanban deve assumere la visione d'insieme, quindi, come primo passo, deve visualizzare il suo flusso di lavoro e riconoscere le fasi che lo compongono per creare una Kanban Board che le rappresenti.

Il flusso è dato dal numero di feature che riescono a essere elaborate dal sistema.

Quando il team trova la procedura ottimale per il rilascio del software e per la ricezione di una quantità adeguata di feedback, allora è riuscito a massimizzare il flusso. Kanban rivolge molta attenzione alla massimizzazione del flusso e al coinvolgimento del team in questo processo poiché quando il flusso aumenta, diminuiscono la frustrazione e i ritardi. Inoltre, il morale del team trova giovamento quando vede che i risultati ottenuti rispettano le aspettative e i tempi stimati.

Nella gestione dei rilasci è possibile vedere un esempio di cosa significhi avere una visione d'insieme. Infatti, la quantità di rilasci desiderata non deve essere calibrata solo in base al flusso, perché se il progetto riceve pochi feedback avrà un percorso di sviluppo poco lineare e con drastici scossoni per riassestarlo in base alle richieste del cliente. Invece un progetto che riceve molti feedback può effettuare piccoli aggiustamenti e convergere fin da subito alla soluzione desiderata dal cliente [3].

Lo stesso concetto può essere visto dal lato della quantità dei rilasci: pochi rilasci ma che introducono molte feature possono portare il progetto a deviare rispetto all'idea del cliente e devono, quindi, essere rivisti per tornare sulla strada corretta; invece, progetti che effettuano tanti piccoli rilasci sono sempre in grado di intervenire appena il cliente riscontra una differenza rispetto alle sue aspettative, il progetto in questo caso richiede piccoli interventi per ritornare sul tracciato corretto [3].

Affinché il team sia in grado di effettuare dei rilasci rapidi è necessario che ogni membro del team sappia sempre cosa fare per massimizzare il valore che viene aggiunto al progetto; questo obiettivo può essere raggiunto quando il manager assegna a ciascun sviluppatore il lavoro oppure quando il team è auto-organizzato, ma nel caso di ambienti che evolvono in fretta solo la seconda opzione diventa percorribile [3].

Il principio per cui è meglio fare rilasci il più frequentemente possibile è complementare al principio di rimandare le decisioni all'ultimo momento utile poiché più si è veloci a effettuare i rilasci più si possono ritardare le decisioni; infatti, le opzioni rimangono disponibili finché non si hanno maggiori informazioni per effettuare la scelta in modo più consapevole [3].

Effettuare le scelte all'ultimo momento utile è un altro aspetto caratteristico di Kanban. L'applicazione di questo principio comporta il fatto che il team deve passare da uno sviluppo di tipo sequenziale a uno di tipo concorrenziale, cioè cominciare l'implementazione dalle feature con il più alto valore aggiunto, quelle che interessano maggiormente il cliente, non appena il design di più alto livello è completato. Potrebbe sembrare controintuitivo cominciare lo sviluppo prima di aver pensato a un design completo del software, ma è possibile vedere lo sviluppo concorrenziale in modo differente: il team adotta un approccio di tipo esplorativo, nel senso che prova varie opzioni e solo quando è convinto della validità di una soluzione la preferisce

alle altre [3].

Questo approccio allo sviluppo permette di evitare i costi dovuti a un design errato perché, in un ambiente dove i requisiti cambiano frequentemente, tutte le decisioni vengono rimandate continuamente, mantenendo però aperte le strade per tutte le opzioni possibili. Quando il cambiamento è inevitabile, lo sviluppo di tipo concorrente riduce i tempi di consegna e i costi generali senza intaccare la bontà del prodotto finale. Purtroppo per poter applicare lo sviluppo concorrente il team deve avere degli sviluppatori esperti nel dominio che possano prevedere quali parti del design siano più critiche [3].

È importante notare, però, che rimandare le scelte all'ultimo momento utile significa prendere una decisione finché ci sono alternative, altrimenti la scelta sarà obbligata e difficilmente sarà ottimale.

Il metodo Kanban è molto semplice e poco strutturato; quindi, è molto flessibile e facilmente adattabile, ma fornisce poche linee guida per il management, che quindi potrebbe essere riluttante ad applicarlo. Per questo motivo è importante per il team misurare le proprie performance e avere strumenti per comunicare i risultati ottenuti, in modo da convincere il management dell'efficacia del metodo scelto.

2.1.2 I ruoli

Kanban è un metodo che non prevede l'assegnamento di nuovi ruoli come avviene in Scrum, anche se ci sono dei ruoli che tendono a emergere nel corso del tempo:

- Kanban Coach
- Service Delivery Manager
- Service Request Manager

- Development Team

Il Kanban Coach ha il compito di controllare che il team adotti il metodo Kanban con l'approccio corretto. In caso contrario il Kanban Coach deve educare il team ai valori, ai principi e alle pratiche Kanban. A volte questo ruolo viene integrato con quello del Service Delivery Manager, se la persona individuata per quel ruolo ha già dimestichezza con i metodi Lean.

Il Service Delivery Manager (SDM) è anche noto con il nome di "flow manager" o "flow master". Questo ruolo non prevede la creazione di una nuova posizione, ma viene assegnato a un membro del team, che, quindi, avrà delle nuove responsabilità che si aggiungono a quelle precedenti.
Il compito del SDM è simile, per certi aspetti, a quello dello Scrum Master. Deve controllare che il flusso si mantenga stabile e se ci sono feature che rimangono bloccate per troppo tempo interviene per trovare una soluzione con il resto del team. Inoltre deve verificare che i membri del team rispettino le Policies e chiarire quali siano le priorità per il cliente, anche se non è un suo compito quello di stabilire quali siano le priorità. Un'altra attività che deve essere svolta da questa figura è quella di cercare continuamente nuove strategie per incrementare la capacità del flusso e trasmettere questa attitudine agli altri membri del team.

Il Service Request Manager (SRM) è un ruolo simile a quello del Product Owner. Il suo compito è di comprendere i bisogni e le richieste del cliente con lo scopo di determinare quali feature hanno maggiore priorità e quali meno. Inoltre, si occupa di gestire il rischio e le Policies del sistema ed interviene nelle fasi decisionali per indirizzare il team.
Anche questa figura, come il SDM, non comporta l'introduzione di nuovi membri nel team, ma l'assegnazione di nuove responsabilità a un membro del team.

Il Development Team è costituito da un gruppo di sviluppatori di grandez-

za variabile; solitamente viene individuata la dimensione di 25 sviluppatori come la grandezza ideale, ma un team può arrivare a comprenderne fino a 40.

I ruoli di SDM e SRM non sono obbligatori per adottare Kanban, poiché sono stati individuati per semplificare la fase di transizione verso il metodo Kanban. Infatti, solo recentemente sono emerse queste figure; nella formulazione di Kanban da parte di Anderson non vi è alcun riferimento ad esse.

2.2 Strumenti per implementare Kanban

Il fulcro di questo metodo risiede nella Kanban board; costruirne una è un procedimento molto semplice: si divide una lavagna in tre colonne, ognuna denotata da una etichetta: To-Do, In-Process, Done. Mano a mano che vengono individuati i requisiti, questi vengono scritti su dei Post-It e collocati all'interno della prima colonna (To-Do); quando uno sviluppatore si fa carico di implementare un requisito questo viene spostato dalla colonna To-Do a In-Process, per poi essere posizionato all'interno della colonna Done una volta completato.

Anche se inizialmente la Kanban Board risulta simile alla Scrum Task Board, una volta individuate le fasi del processo di sviluppo la colonna In-Process viene suddivisa in varie colonne per rappresentare al meglio le procedure e le dipendenze presenti nel processo.

A volte individuare le fasi che compongono un processo non è semplice, per riuscire a riconoscere i diversi passaggi di cui è composto lo sviluppo di una feature, Kanban si avvale di uno strumento comune ai metodi Lean: la Stream Value Map. Quest'ultimo è uno strumento visuale che rappresenta il percorso effettuato da una feature all'interno del sistema, confrontando le Stream Value Map di alcune feature è possibile trovare dei fattori comuni che caratterizzano il processo di sviluppo, queste fasi ricorrenti costituiscono un

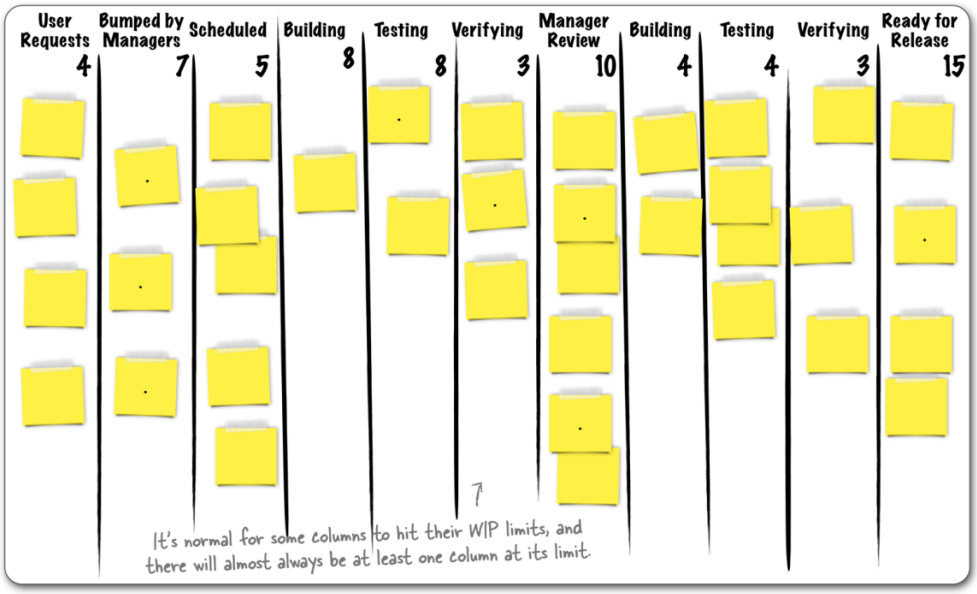

Figura 2.1: Kanban board [2]

buon punto di partenza per riuscire a comprendere la struttura del processo di sviluppo.

La costruzione della Value Steam Map avviene tracciando il percorso effettuato da una feature durante tutto il processo di sviluppo e riportando l'alternanza di fasi attive e passive su un foglio di carta: le fasi attive corrispondono al tempo in cui la feature viene effettivamente implementata, mentre le fasi passive corrispondono a momenti in cui la feature resta in attesa prima di passare alla fase successiva dello sviluppo.

La Value Stream Map è uno strumento molto efficace non solo per distinguere le fasi del processo, ma anche per individuare sprechi nel processo di sviluppo, infatti se si riducono i tempi di attesa la feature verrà realizzata in tempi più rapidi. L'ottimizzazione non deve però avvenire a discapito del resto del sistema, è quindi importante valutare quali siano le conseguenze di ogni modifica al processo non solo per lo sviluppo della singola feature, ma anche per le altri componenti del sistema che ne subiscono le conseguen-

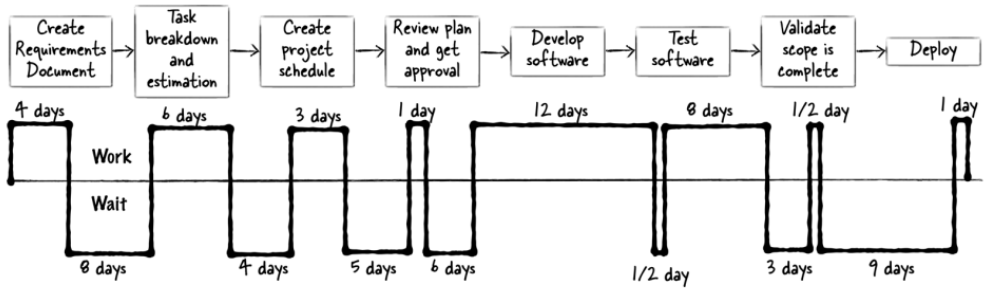

Figura 2.2: Lean Value Stream Map [3]

ze. Questo strumento, nella sua semplicità, diventa molto utile anche per la comunicazione interna, soprattutto con il management, per convincere tutti i soggetti interessati ad effettuare dei cambiamenti perché mostra in modo evidente gli sprechi presenti nel processo di sviluppo.

La Value Stream Map fornisce al team dei validi spunti per fare piccole modifiche al sistema al fine di ottimizzarlo; riuscire a comunicare i risultati ottenuti da questi cambiamenti a tutte le figure interessate è espressione del pensiero Lean: Amplifying Learning è un punto focale del pensiero Lean che corrisponde alla misurazione dei risultati derivanti dalle modifiche al processo e alla loro comunicazione a tutti i membri del team, per renderne noti gli effetti positivi e negativi. Inoltre, il principio Amplify Learning trova la sua applicazione all'interno di Kanban nel rispetto dei ruoli e delle responsabilità già esistenti; infatti, la diffusione dei risultati apportati è utile a tutti i livelli della gerarchia aziendale per poter valutare pienamente gli effetti non solo sul reparto interessato ma anche sui reparti che collaborano con esso.

La Kanban Board non deve necessariamente essere un elemento fisico, ma può anche essere virtuale (esistono numerosi software che la implementano). Ogni team di sviluppo può avere le sue preferenze su quale strumento usare: una lavagna fisica costituisce un punto di incontro naturale tra gli sviluppatori e favorisce la comunicazione faccia a faccia, al contrario una lavagna

virtuale non agevola la comunicazione, ma rende più semplice l'integrazione con altri servizi. Il team sceglie in base alle proprie priorità, ma poiché la comunicazione è un principio fondamentale sia dei metodi Agili che dei metodi Lean, in genearale, è preferibile l'uso di una lavagna fisica.

La Kanban Board ha un'altra differenza rispetto alla Scrum Task Board: ogni fase dello sviluppo ha un limite al numero massimo di requisiti che può ospitare.

Il limite alla quantità di requisiti, che da ora in poi verrà chiamato limite WIP (Work-In-Process), comporta il fatto che quando un requisito non avanza esso rimane valido per il conteggio, se la situazione si ripete per più requisiti il team di sviluppo viene spinto a cooperare per sbloccare la situazione e permettere a nuovi requisiti di entrare in fase di WIP; l'obiettivo è quello di mantenere il flusso sempre attivo.

Molto spesso i team che usano Kanban vengono suddivisi in team funzionali e stabilire quale sia il limite WIP corretto per ogni fase rispetto al numero di sviluppatori non esiste una formula precisa, solitamente si tende a valutare due o tre requisiti per sviluppatore, ma non esiste una soluzione universale e ogni team dovrebbe aggiustare il parametro in modo empirico durante lo sviluppo. In generale è possibile dire che il limite dovrebbe essere il più piccolo possibile, ma ampio abbastanza da assorbire la quantità di nuovi requisiti che vengono inseriti nel sistema [7].

Un altro obiettivo del metodo Kanban è quello di effettuare il rilascio del software il più velocemente possibile. La rapidità con cui si sviluppa una feature e ne si effettua il rilascio è un aspetto sempre più importante nel mercato informatico odierno. Riuscire ad effettuare rilasci molto frequenti è importante per poter ritardare le scelte, infatti ad ogni rilascio il cliente fornisce un feedback al team di sviluppo e più feedback si ricevono più si è

in grado di valutare le aspettative del cliente e il suo grado di soddisfazione, in modo da comprendere in quale direzione rivolgere lo sviluppo. In termini pratici, riuscire a effettuare rilasci rapidi significa che il team è in grado di rilasciare le feature richieste prima che il cliente abbia il tempo di cambiare idea [3].

La rapidità dei rilasci e il fatto di ritardare le decisioni all'ultimo momento utile rende difficile per il management poter gestire le attività degli sviluppatori, quindi i team che usano Kanban devono essere auto-organizzati.

Quando si vuole misurare l'efficacia del processo di sviluppo viene spesso valutato il tempo di consegna (lead time) per ogni feature, che viene calcolato in questo modo: quando un cliente fa una richiesta, si registra la data come data di inizio, mentre quando viene rilasciata una versione del software che contiene la feature richiesta allora si registra la data di completamento della feature; la quantità di tempo che intercorre tra la data di fine e di inizio è il tempo di consegna [2]. Se si misura il tempo di consegna per ogni feature e ne si fa una media, il team può stimare più facilmente quanto tempo è necessario per sviluppare una feature. È possibile immaginare di raffinare ulteriormente il metodo dividendo le feature in classi di complessità per poter determinare con maggior precisione le stime per feature di complessità diverse.

Nel metodo Kanban, e più in generale nei metodi Lean, è molto importante fare delle misurazioni. In primo luogo la raccolta dei dati aiuta a comprendere il processo di sviluppo e a suddividerlo nelle fasi che lo compongono. Inoltre, poter fornire al management dati concreti che illustrano problematiche o che rendono evidenti i miglioramenti ottenuti può essere un elemento di forte motivazione per entrambe le parti in causa a fare dei cambiamenti, visto che molti esperimenti empirici hanno dimostrato che le persone tendono a evitare i cambiamenti [2].

Altri strumenti di cui Kanban dispone vengono dal mondo Lean: sistemi di tipo pull, teoria delle code e il costo del ritardo (cost of delay).

I sistemi di tipo pull si contrappongo ai più classici sistemi di tipo push. Un sistema che usi la metodologia Waterfall è un sistema di tipo push perché lo sviluppatore riceve i compiti da svolgere dal suo superiore, è quest'ultimo che "spinge" i task verso gli sviluppatori. Invece, un sistema di tipo pull è un sistema in cui ogni sviluppatore sceglie su quali task lavorare, è quindi lo sviluppatore che "trascina" verso di sé il task. Un altro aspetto interessante dei sistemi pull è che lasciano il manager fuori dal ciclo decisionale, poiché i team sono auto-gestiti, quindi il manager svolge più il ruolo di coach che quello di direttore [3].

La teoria delle code è studiata per evitare che gli sviluppatori siano oberati di lavoro e riescano, quindi, a trovare il tempo per terminare i propri incarichi. Le feature da implementare costituiscono una coda e quando uno sviluppatore ha completato i task di cui si era fatto carico può prendere un elemento dalla coda, solitamente la coda è ordinata in base alla priorità o al tempo di giacenza dei compiti all'interno della coda stessa. Nei metodi Lean si è visto che rendendo pubbliche le code ci sono miglioramenti nei tempi di rilascio del software; inoltre, tenere in considerazione gli effetti che si avranno sulle code durante le fasi di decision-making aiuta i team a mantenere una buona velocità di rilascio [2].

Per misurare come il team rilascia le feature è possibile usare la WIP Area Chart. Quest'ultimo è un diagramma che illustra il flusso delle feature attraverso il processo di sviluppo. Lo scopo della WIP Area Chart è di mostrare uno storico delle feature di cui è stata cominciata l'implementazione; da questo grafico è possibile vedere quali feature erano in stato di lavorazione e in quali giorni, inoltre è possibile notare quali fasi sono state attraversate

da ogni feature. è importante notare che la WIP Area Chart contiene solo feature e non i singoli task che le compongono. Una WIP Area Chart viene costruita in questo modo: l'asse delle ascisse è composto dalle date, invece l'asse delle ordinate conta le feature in lavorazione. Inoltre, a ogni fase individuata nella Value Stream Map corrisponde una linea nella WIP Area Chart; queste linee individuano delle aree corrispondenti alle fasi della Value Stream Map. Inizialmente la WIP Area Chart non contiene feature quindi si parte dall'origine degli assi (giorno: 0, feature in lavorazione: 0), successivamente vengono registrate le feature mano a mano che entrano nelle varie fasi del processo di sviluppo.

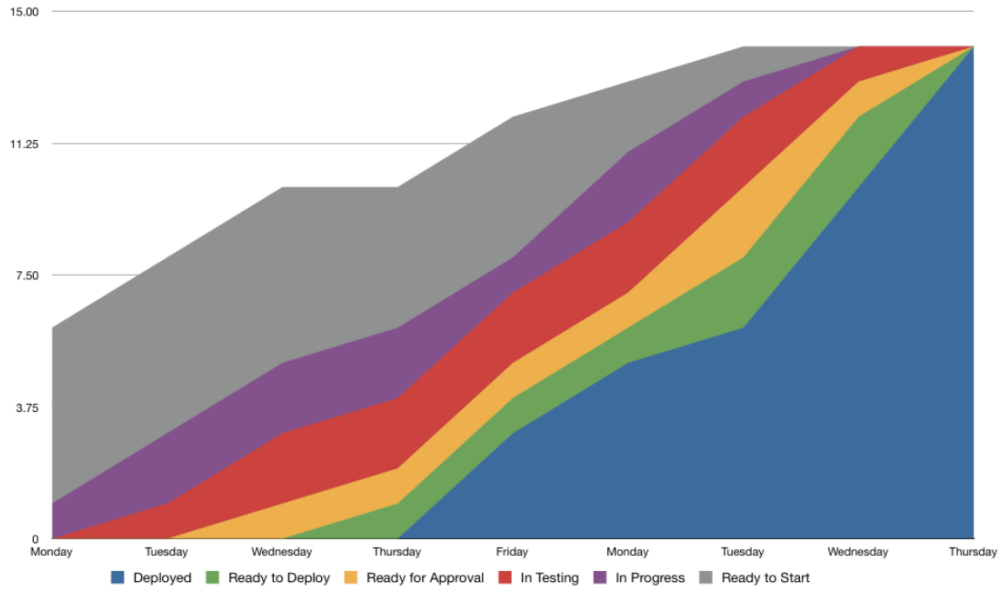

Figura 2.3: Cumulative flow diagram [14]

Quando le feature vengono completate vanno a riempire l'area più a destra del grafico, alla lunga il grafico diventa illegibile e quindi poco utile; un grafico siffatto illustra la crescita della quantità di feature su cui si è lavorato. Invece, se si rimuovono le feature completate il grafico diventa molto utile per visualizzare il flusso e identificare il trend, il risultato è che è possibile notare a colpo d'occhio se il flusso procede o se sta incontrando intoppi che

lo porterebbo a fermarsi.

La teoria dei vincoli afferma che, un vincolo specifico può limitare la quantità di lavoro che il flusso riesce a gestire. Quando viene individuato un collo di bottiglia è possibile modificare il vincolo che lo determina per osservare come reagisce il sistema, se viene individuato un nuovo collo di bottiglia è possibile intervenire anche su questo. L'obiettivo è quello di trovare la combinazione di vincoli che massimizzi la capacità del flusso del sistema.

Quando si creano dei colli bottiglia è possibile agire in vari modi; una modalità è quella di modificare il limite WIP dell'area interessata, quando l'unico effetto che si ottiene è quello di spostare il collo di bottiglia da una fase all'altra del processo è consigliabile intervenire direttamente sul processo; un'altra strada praticabile è quella di inserire dei buffer tra una fase e l'altra, anche il buffer avrà un suo limite WIP al fine di individuare uno stallo se le feature che si trovano nel buffer non riescono ad avanzare verso la fase successiva.

La Kanban Board è uno strumento visuale molto efficace per visualizzare il flusso di lavoro, ma la sua utilità non si limita all'ottimizzazione del processo di sviluppo. Un aspetto che spesso non viene considerato dannoso perché ci si fa l'abitudine è il multitasking: quando gli sviluppatori sono oberati di lavoro e devono continuamente destreggiarsi tra progetti o feature differenti senza potersi concentrare adeguatamente su quello che stanno facendo si perde capacità produttiva.

La Kanban Board permette di individuare la fase del processo in cui avviene più frequentemente lo stallo delle feature e ponendo un limite WIP più stringente si può fare in modo che il team si concentri maggiormente sulla fase in questione per sbloccare più velocemente la situazione. Infatti gli sviluppatori che devono spostare le feature su cui hanno lavorato nella fase che costituisce il collo di bottoglia saranno costretti a non fare avanzare le proprie feature alla fase successiva finché lo stallo non venga risolto. Va no-

tato come l'avanzamento di una feature non sia mai una mossa obbligata, in casi di stallo risulta più conveniente interrompere il lavoro su quella feature e spostarsi su un'altra che abbia la possibilità di avanzare. Questa situazione ci spiega perché i metodi Lean cerchino di ritardare le scelte all'ultimo momento utile: se si fosse deciso fin dal principio a quali feature lavorare e in che tempi si sarebbe rischiato di vedere i propri piani non rispettati, invece lasciando che sia il flusso a dirigere i lavori è possibile prendere ogni volta una decisione più consapevole perché si hanno più informazioni con cui poter fare le proprie valutazioni. Prima di cominciare l'implementazione del progetto è difficile prevedere quale sarà la fase del processo di sviluppo in cui avverrà lo stallo, invece, una volta che il progetto è in stato di sviluppo, è più semplice individuare quali fasi sono più critiche e problematiche perché è l'evidenza empirica a confermarlo.

Tutti questi strumenti hanno lo scopo di rendere il processo di sviluppo il più fluido ed efficiente possibile con lo scopo di poter fare rilasci il più frequentemente possibile, idealmente si vorrebbe un sistema in grado di fare rilasci on-demand per poter fornire le feature ai propri clienti in tempi quanto più rapidi possibili. Inoltre i rilasci frequenti hanno il vantaggio di poter avere un feedback veloce da parte del cliente, mentre un feedback ritardato di alcuni mesi potrebbe implicare lo stravolgimento di più feature andando a creare spreco perché gli sviluppatori devono tornare a lavorare su parti di software che venivano considerate già terminate e ormai consolidate nel progetto. Il limite WIP costituisce il migliore strumento di controllo del flusso e come tale permette di controllare anche i tempi dei rilasci. Più il limite WIP dell'ultima fase è basso, più spesso ci saranno rilasci; se invece si ricevono troppi feedback si alza il limite WIP per fare meno rilasci. A volte ricevere troppe informazioni costituisce un problema perché se non si ha il tempo per elaborarle ci si trova ad oberare di lavoro alcuni membri del team di sviluppo che sono incaricati di raccoglierle e introdurre nel sistema le modifiche opportune [2].

Si è visto come viene gestito e indirizzato il flusso di lavoro, ma non è stato ancora individuato un modo per misurare la sua capacità e questo aspetto è importante per poter valutare se i cambiamenti che vengono introdotti nel sistema comportino più vantaggi che svantaggi. Lo strumento per misurare la capacità del flusso è il Cumulative Flow Diagram (CFD).

Il CFD è simile al WIP Area Chart, ma con la differenza che invece di eliminare i requisiti completati, questi vengono mantenuti nel diagramma. Inoltre il CFD, invece di suddividere le feature in base allo stato dello sviluppo, suddivide le feature in base alla fase del processo di sviluppo in cui si trovano, quindi le feature seguono lo stesso raggruppamento che hanno sulla Kanban Board.

Il CFD può anche contenere delle linee che rappresentano i tempi medi di consegna, la media di feature che vengono aggiunte ogni giorno e la quantità di feature totali presenti nel processo produttivo; queste informazioni aggiuntive possono essere utili per individuare situazioni di pericolo per il flusso di lavoro. Mentre la Kanban Board permette di individuare i colli di bottiglia su base quotidiana, il CFD fornisce uno sguardo d'insieme e permette di visualizzare il sistema nella sua interezza e di individuare i problemi che si generano sul lungo periodo; la combinazione di entrambi questi strumenti è fondamentale per capire come gestire il processo di sviluppo e quali cambiamenti siano effettivamente vantaggiosi per la buona riuscita del progetto.

Per costruire il CFD si parte dalla WIP Area Chart, ma invece di usare i dati forniti dal Value Stream Map si usano i dati forniti dalla Kanban Board (si contano le feature in ogni colonna). Vengono poi aggiunti ai dati così ottenuti quelli riguardanti la quantità di feature che vengono aggiunte quotidianamente al progetto: questo dato corrisponde al numero di feature che vengono aggiunte ogni giorno nella prima colonna della Kanban Board.

Un'altra informazione che viene spesso aggiunta è quella riguardante la quantità di feature attualmente in sviluppo: corrisponde alla somma delle feature presenti in tutte le colonne della Kanban Board.
I dati riguardanti la quantità di feature che vengono inserite nel progetto e quante feature in totale siano presenti sono funzionali alla comprensione di quanto il sistema sia stabile: se le linee ricavate da questi dati sono orizzontali allora il sistema è stabile, altrimenti il sistema è in fase di cambiamento ed è necessario intervenire modificando i limiti WIP per correggerne l'andamento [2].

Quando un sistema è stabile rispetta la legge di Little, un teorema appartenente alla teoria delle code, che viene espressa dall'equazione

$$L = \lambda W$$

. Il significato di questa equivalenza è che, in un sistema stabile, la quantità media di feature nel sistema (L) è uguale al tasso medio di feature in entrata (λ) moltiplicato per il tempo medio di rilascio delle feature (W). La legge di Little è molto utile per calcolare il tempo medio di rilascio perché gli altri due parametri sono facilmente ottenibili, mentre il tempo medio di rilascio delle feature può essere più complicato da seguire e avere una formula per ricavarlo può far risparmiare tempo. Inoltre questa formula garantisce che il tempo di medio dei rilasci dipende esclusivamente dalla quantità di feature introdotte nel sistema e dalla quantità di feature presenti in esso, quindi se si vuole ridurre il tempo medio dei rilasci è sufficiente ridurre la quantità di feature in entrata nel sistema, infatti $W = L/\lambda$ [2].

Le potenzialità di Kanban sono ben espresse dagli strumenti illustrati: la capacità di poter visualizzare il flusso, poterlo misurare e, una volta che il sistema è stabile, poterlo controllare per adattare i tempi dei rilasci alle proprie esigenze sono dei validi motivi per avvalersi di questo metodo.

Lo sviluppo di un software è un'attività che difficilmente è possibile programmare in modo preciso, diventa quindi indispensabile fornire dei margini temporali per evitare di oberare di lavoro il team di sviluppo.
La possibilità per gli sviluppatori di avere sempre il tempo necessario per fare un buon lavoro è di vitale importanza per evitare che vengano introdotte feature implementate in maniera frettolosa nel sistema, perché significherebbe introdurre dei problemi che andrebbero risolti in futuro.
Un metodo per creare del tempo extra utilizzabile dagli sviluppatori è quello di gestire i limiti WIP in modo da evitare che venga accumulato troppo lavoro in una singola fase del processo. Anche in Kanban, come nei metodi iterativi, i rilasci vengono solitamente stabiliti su scadenze fisse, ma non viene posto alcun vincolo su quali feature debbano essere presenti al momento del rilascio, invece, deve essere il flusso dello sviluppo a generare le feature in tempo per il rilascio; se il sistema è stato ottimizzato a dovere l'obiettivo viene raggiunto come preventivato. Inoltre, i limiti WIP garantiscono che anche le feature extra che vengono introdotte nel sistema non vadano a inficiare il lavoro del team di sviluppo perché i limiti WIP fungono da tornelli e impediscono che le feature appena introdotte nel sistema vadano a oberare di lavoro parti del sistema già sotto sforzo; per questo motivo ogni colonna della Kanban Board deve avere il suo limite WIP [2].

Kanban prevede la pratica di scrivere le fasi del processo di sviluppo per descrivere come lavora il team di sviluppo e mostrarlo a chiunque sia coinvolto nel processo. Quindi, i team Kanban non devono scrivere lunghe documentazioni o creare dei Wiki per esplicitare le Policy perché queste vengono espresse anche solo con i limti WIP. In aggiunta il team può decidere di descrivere meglio alcune Policy, per esempio può specificare quando una feature può considerarsi completa nella sua implementazione, oppure può aggiungere dei criteri aggiuntivi in alcune colonne della Kanban Board [2].

Kanban guarda all'intero sistema come un'unica entità invece di provare

a gestirne ogni singola attività perché accetta il fatto che le singole attività di sviluppo possano aver delle variazioni rispetto a quanto preventivato, mentre il sistema tenderà a essere predicibile nel suo modo di operare [2].

2.3 Le metriche di Kanban

Poiché Kanban è un metodo rivolto a migliorare un processo di sviluppo comprende anche delle metriche per poter valutare se l'applicazione di questo metodo abbia apportato delle migliorie al sistema.

Le metriche più diffuse sono:

- Cycle time / production Lead Time
- Tact Time / Takt Time
- Customer Lead Time
- Throughput
- Work in Progress (WIP)
- Le code
- Blockers
- La cadenza

Il Lead Time è il tempo richiesto per la realizzazione di una feature calcolato partendo da quando essa viene richiesta dal cliente fino al suo rilascio. Il Cycle Time, invece, corrisponde al tempo che è stato effettivamente dedicato per lo sviluppo di una feature ed è uguale alla somma dei tempi richiesti da ogni fase del processo di sviluppo. Quindi il Lead Time può essere interpretato come una misurazione dell'intero ciclo di sviluppo di una feature, mentre il Cycle Time distingue il lavoro sulla feature in base alle fasi dello

sviluppo. Questa metrica permette di stimare quanto il team è rapido nei suoi interventi e quanto è veloce a effettuare rilasci quando le priorità vengono modificate.

Per poter effettuare questa valutazione è necessario tenere traccia di tut-

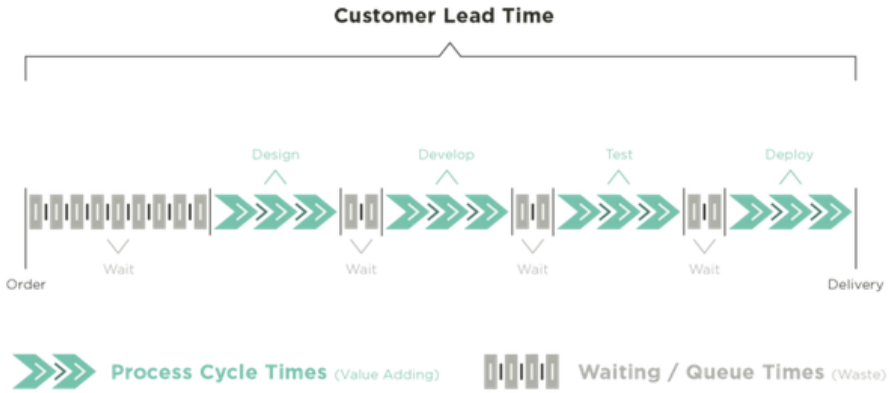

Figura 2.4: Customer Lead Time vs Process Cycle Times [14]

ti i momenti in cui una feature entra ed esce da una fase dello sviluppo. Il vantaggio apportato da queste metriche è che rendono possibile valutare quanto tempo viene impiegato dal team per sviluppare una feature con una certa priorità e questi dati possono essere usati per raffinare le previsioni sulle prossime feature da implementare.

Poiché Kanban fa largo uso di strumenti visivi per evidenziare i dati raccolti, fornisce anche un diagramma per visualizzare il Lead Time e il Cycle Time noto con il nome di Scatter Diagram. Quest'ultimo viene usato al fine di evidenziare dei trend, poiché l'idea alla base di questo strumento è quella di permettere di individuare tempestivamente dei trend negativi la precisione dei dati non ha altrettanta rilevanza quanto il loro costante aggiornamento. Questo diagramma viene costruito indicando sull'asse delle ascisse le feature sviluppate in ordine temporale e sull'asse delle ordinate lo sforzo che è stato necessario per implementarle.

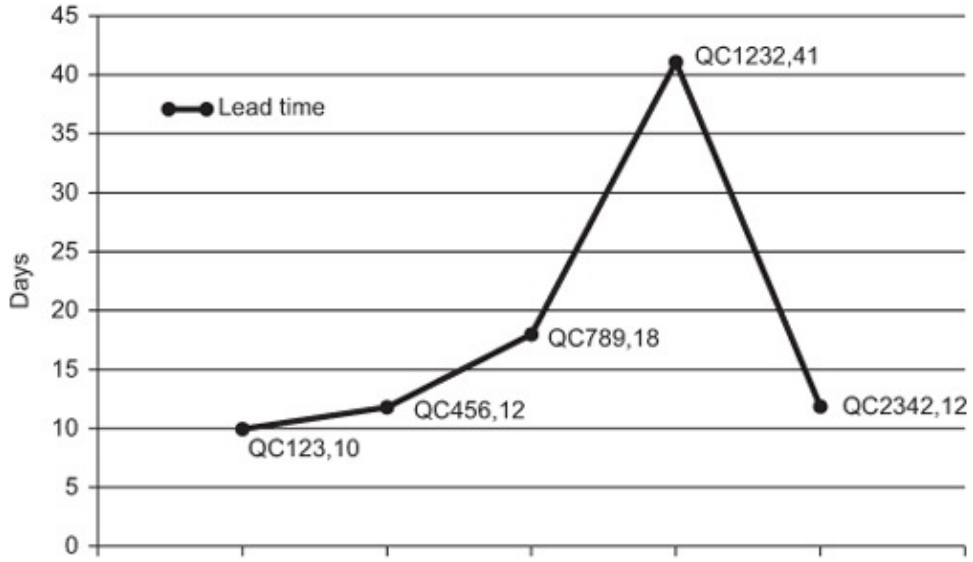

Figura 2.5: Scatter diagram [15]

Il Takt Time è una metrica che misura la frequenza dei rilasci delle nuove feature. Viene calcolato dividendo il Cycle Time medio per la quantità media di attività in fase di sviluppo, quindi misura il throughput del Development Team e permette di stimare se le feature ancora nel Backlog potranno essere completate entro la scadenza prefissata.

Il Costumer Lead Time è simile al Takt Time, ma comincia a misurare il tempo da quando il cliente ha richiesto la feature. La differenza con il Takt Time è che se un'attività rimane a lungo nel backlog, essa non incide sul Cycle Time e non influisce sul Takt Time. Quindi il Costumer Lead Time valuta la capacità del sistema nel suo complesso di reagire ai cambiamenti.

Il Throughput è definito come la media di feature lavorate per unità di tempo ed è una misura della produttività. Questa metrica permette al team di comprendere la sua efficienza nel rilasciare feature. L'importanza di questo dato risiede nella sua capacità di catturare i miglioramenti nei tempi dei rilasci o, viceversa, il loro peggioramento. In realtà, non è molto utile per

fare previsioni future, ma diventa interessante quando viene combinato con altre metriche al fine di individuare un trend.

Il Work In Progress (WIP) corrisponde al numero di feature in stato di sviluppo. Questo dato non è molto interessante di per sé, ma diventa molto utile quando viene combinato con altri dati, come avviene, per esempio, nel Takt Time e nel Customer Lead Time. Inoltre il WIP compare anche nella legge di Little, un teorema che ha una forte rilevanza in Kanban, ed è quindi una metrica da tenere sotto stretto controllo. Infatti, il limite WIP se usato correttamente riduce il multitasking e aumenta il throughput.

Le code corrispondono a quei momenti del processo di sviluppo dove una feature resta in attesa di passare da una fase all'altra. Tracciare questi tempi "morti" non è sempre facile, a questo scopo vengono spesso adottati dei software che li stimano in relazione all'andamento dei limiti WIP. Poiché le attività trascorrono molto tempo all'interno delle code è importante capire come questo aspetto influisca sul resto del sistema; inoltre, più si riducono i tempi di attesa delle feature più si riducono i tempi dei rilasci.
A questo scopo è possibile adottare un Efficiency Diagram; quest'ultimo misura la differenza tra la quantità di attività in lavorazione (WIP) e quelle presenti nelle code. In questo modo si rende evidente quando le attività presenti nelle code crescono in rapporto a quelle in fase di sviluppo rendendo, quindi, possibile individuare quali code stiano accumulando attività in quantità eccessiva [14].

Con Blockers si intende la quantità di feature rimaste bloccate (in una coda), esse sono un ostacolo per il flusso di sviluppo. Le attività che non possono avanzare sono facili da individuare perché generano uno stallo nel processo di sviluppo; affinché si verifichi questa situazione devono esistere delle dipendenze esterne non soddisfatte o, più in generale, delle condizioni fallimentari. Questa metrica può essere ulteriormente raffinata conteggiato separatamente le feature bloccate in base alla fase dello sviluppo raggiunto.

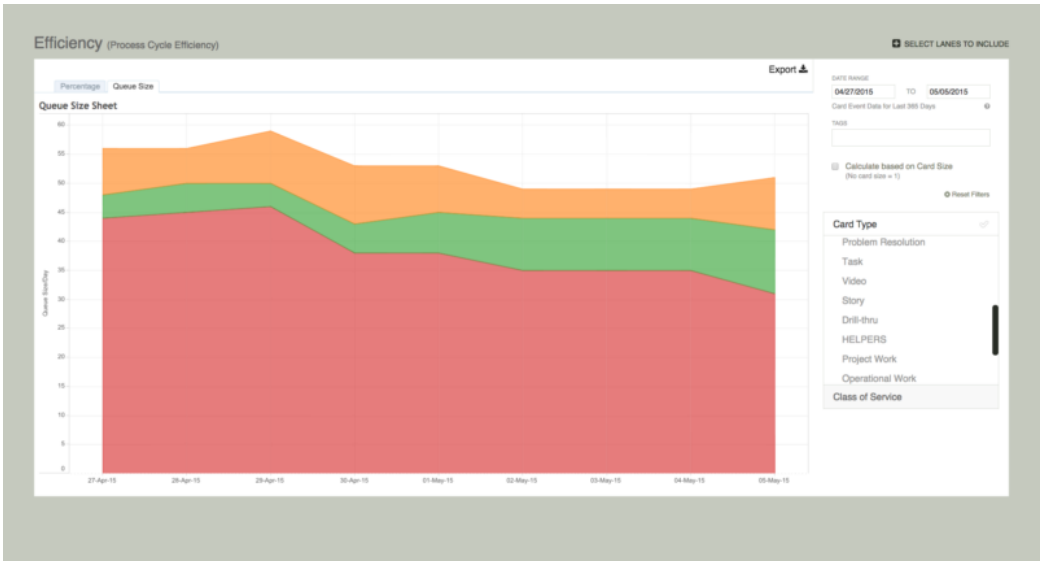

Figura 2.6: Efficiency diagram [14]

La quantità di feature rimaste bloccate nel processo di sviluppo devono essere tenute sotto controllo per evitare di incorrere continuamente in situazioni di stallo; inoltre, questa metrica permette di valutare la qualità del software tra una fase e l'altra del processo di sviluppo.

La cadenza è una metrica che descrive un percorso per ottenere un sistema affidabile. Poiché Kanban non prevede delle iterazioni, potrebbe sembrare difficile stabilire qualsiasi forma di stima riguardo al completamento delle feature. In realtà, misurando il throughput del sistema è possibile effettuare delle stime affidabili; inoltre, è possibile fare uso di diagrammi come il Cumulative Flow Diagram per ottenere una rappresentazione visiva.

Capitolo 3

Scrumban

Il metodo Scrumban è stato introdotto da Corey Ladas nel suo libro "Scrumban: Essays on Kanban Systems for Lean Software Development"; in questo testo l'autore afferma di aver pensato Scrumban come un percorso evolutivo per permettere a un team di sviluppatori di cambiare il proprio metodo da Scrum a Kanban, o più in generale ai metodi Lean. Il passaggio a un nuovo metodo non è mai semplice e questi cambiamenti avvengono meglio se non vengono effettuati in modo drastico, ma introdotti gradualmente. In questo contesto Scrumban si propone come una valida via di mezzo tra i due metodi per poter abbandonare gli strumenti e i principi di Scrum e adottare quelli di Kanban.

Scrumban privilegia lo sviluppo just-in-time e riduce i tempi di rilascio, queste caratteristiche lo rendono vantaggioso in alcune situazioni in cui il team trova Scrum troppo rigido rispetto alla flessibilità richiesta dai compiti da svolgere.

Nel caso di progetti che hanno già terminato la fase di sviluppo e sono ormai in fase di manutenzione, Scrumban può essere usato per gestire una situazione dove gli interventi richiesti sono di tipo event-driven (come la risoluzione di bug) oppure se è necessario fornire un supporto ai clienti;

ricordando che la fase di manutenzione corrisponde a circa metà del processo di sviluppo del progetto. Questo avviene perché il software richiede cambiamenti dovuti all'evoluzione dell'ambiente in cui opera (aggiornamento del sistema operativo o delle librerie o del database sono tra i motivi più comuni); quando il software non viene più aggiornato può essere considerato deprecato, se questo avviene perché il software è troppo difficile da modificare allora sono state sprecate delle risorse [3].

> "Trovare e risolvere i problemi di un software dopo il suo rilascio costa 100 volte di più che trovarli e risolverli in fase di design."
> Barry Boehm, 1987

Un altro caso in cui Scrumban può risultare vantaggioso è quando un progetto è nelle fasi di hardening e packaging, poiché in questi stadi dello sviluppo non avviene l'introduzione di nuove feature.

Esistono, però, anche casi in cui un progetto è in fase di sviluppo, ma un metodo meno rigido e con una gestione più lasca dei tempi di sviluppo può aiutare il team a recuperare la situazione e riportare il progetto sui binari: per esempio quando un progetto riscontra molti più errori rispetto a quanto preventivato o ha la necessità di introdurre continuamente nuove User Story nel processo di sviluppo; oppure quando uno Scrum Team ha problemi a gestire il flusso di lavoro o a sfruttare correttamente le risorse disponibili, oppure il processo di sviluppo riscontra spesso degli stalli.

Un'altra situazione in cui è possibile applicare Scrumban è quello di gestire le fasi che precedono e seguono uno Sprint con questo metodo, mentre durante lo Sprint si continua a usare Scrum [5].

Per capire se l'adozione di un metodo meno rigido sia utile al progetto è possibile osservare dove avvengono più spesso dei cambiamenti all'interno di

uno sviluppo iterativo, questo è un valido punto di partenza per capire dove il sistema richieda maggiore flessibilità. Inoltre, se alcune aree del software richiedono continue modifiche nel corso dello sviluppo, molto probabilmente le stesse aree saranno oggetto di interventi in fase di manutenzione [3].

3.1 Spiegare Kanban a un Team Scrum

Un primo step necessario per introdurre il metodo Kanban a uno Scrum Team è la comprensione di cosa sia un kanban: una parola giapponese che può essere tradotta col termine "segnaposto". Quindi, un kanban è una scheda con la descrizione di un lavoro da svolgere e rappresenta una parte della capacità produttiva del team; in questo senso può essere interpretato come un mezzo di scambio per beni e servizi offerti dagli sviluppatori. Questa analogia tra la moneta e i kanban ci permette di capirne il valore: come la moneta deve essere controllata, allo stesso modo la quantità di kanban presenti nel sistema deve essere regolato per mantenerlo stabile.
Prendendo spunto dall'analogia precedente possiamo dire che una scheda senza limiti non è un kanban, esattamente come la fotocopia di una banconota non è una moneta.
Quindi, la forza di Kanban non risiede nelle schede che descrivono le User Story, ma nei limiti che vengono imposti ai kanban [1].

Questo significa che un kanban serve per indicare la richiesta di un lavoro (domanda), ma anche il permesso di poterlo compiere (offerta); il fatto che una feature oltre a essere richiesta deve anche avere il "permesso" di essere implementata è uno dei concetti del pensiero Lean più difficili da introdurre all'interno di un team, ma questo passaggio è importante perché il rispetto dei limiti è essenziale per applicare correttamente Scrumban.

L'uso corretto dei limiti è fondamentale per controllare la quantità di

lavoro di cui il team si fa carico e per riuscire a rilasciare nuove feature velocemente. Quello che viene misurato è il numero di Use Case da implementare, ognuno di essi deve essere testabile e tracciabile, al fine di poter verificare che soddisfi le richieste del cliente prima del suo rilascio. Gli Use Case sono descrizioni sul funzionamento del software in base alle aspettative del cliente e non contengono informazioni riguardo al design dell'implementazione. Essi possono anche contenere riferimenti ad altri Use Case, in questi casi è necessario dividere quelli di grandi dimensioni in Use Case atomici, cioé che non contengono riferimenti ad altri Use Case; uno Use Case atomico corrisponde a una feature ed è, quindi, candidato per passare alla fase di sviluppo [1].

Poiché Scrumban eredita le iterazioni da Scrum è necessario sincronizzare lo sviluppo delle feature per poterle rilasciare al termine dello Sprint, ma la sincronizzazione è possibile solo quando il tempo dedicato all'iterazione ha una durata uguale o superiore al più lento tra gli stadi del processo di sviluppo.

La sincronizzazione ha il vantaggio di garantire un maggior controllo sul flusso di lavoro poiché limita il numero di feature in ingresso, permette di standardizzare le attività, rende il sistema di tipo pull, consente la separazione dei lavori, permette la condivisione delle conoscenze, consente di tenere traccia dello stato delle feature, infine, permette di riceve feedback.

D'altra parte, però, la sincronizzazione comporta degli svantaggi legati all'incapacità di adattamento a variazioni nella quantità di feature in entrata nel sistema, inoltre eventuali variazioni al normale processo di sviluppo causano maggiori ritardi nel passaggio di alcune feature da uno stato all'altro del processo di sviluppo, questo può essere considerato uno spreco di risorse perché le feature avrebbero potuto avere tempi di implementazione più rapidi; inoltre, un eventuale stallo del flusso di lavoro può comportare maggiori costi per recuperare la situazione e riprendere le normali procedure di svi-

luppo.
In definitiva, la sincronizzazione del flusso di lavoro è efficace in situazioni di sviluppo collaudate, ma, se si introducono delle variazioni, il sistema rischia di diventare molto instabile [1].

Quando tutto il lavoro viene svolto attraverso un unico processo di sviluppo si corre il rischio, che, se questo processo si blocca, tutto il flusso di lavoro viene interrotto. Questa situazione può essere evitata facendo uso di più processi di sviluppo che operano parallelamente, riducendo, così, il rischio di mandare in stallo il flusso di lavoro se un singolo processo si blocca. Inoltre, processi di sviluppo paralleli sono utili per incrementare, anche solo temporaneamente, la capacità di sviluppo del sistema [1].

La sincronizzazione può essere migliorata dall'uso di una Kanban Board in cui ogni stadio ha un limite WIP che corrisponde al tempo richiesto dalla fase di sviluppo in proporzione al tempo impiegato per implementare la feature. Poiché questo limite stabilisce quale sia la massima capacità per ogni fase di sviluppo, se uno stadio non è al massimo della sua capacità allora può prendere una feature dalla fase precedente, invece, se uno stadio ha già raggiunto il suo limite allora non può farsi carico di nuove attività, ma può solo attendere che una delle sue feature proceda alla fase successiva. Questa gestione del flusso di lavoro è sicuramente meno suscettibile a variazioni ed è più stabile rispetto alla sincronizzazione; infatti, anche se uno stadio della fase di sviluppo si blocca, ci vorrà del tempo prima che tutto il processo di sviluppo vada in stallo, nel frattempo è sempre possibile intervenire per risolvere la situazione [1].

La sincronizzazione è un primo passo per adottare il metodo Kanban e le sue inefficienze possono essere viste come i buffer di cui si era discusso nella descrizione della Kanban Board; come era già stato notato, i buffer corrispondono a sprechi, ma alcuni sprechi sono più facili da controllare e,

a volte, sostituire uno spreco con un altro può portare migliorie al sistema nel suo complesso. Per la gestione dei buffer è sempre possibile sfruttare il controllo fornito dall'uso dei limiti WIP [1].

Inoltre, i buffer sono utili per gestire più linee del processo di sviluppo poiché queste devono effettuare i passaggi da una fase all'altra dello sviluppo allo stesso tempo. Se si vuole rimuovere la sincronizzazione, è possibile introdurre dei buffer per regolare più processi di sviluppo che operano in parallelo. Poiché i buffer sono molto utili, diventa importante poterli controllare. La misura ideale per un buffer sarebbe che fosse pari a 0 perché ogni buffer rappresenta uno spreco, però se sono necessari per gestire la sincronizzazione, allora la loro misura ideale è di 1. Nel caso un buffer cominci ad accumulare troppe feature in coda, significa che è necessario rivedere i limiti WIP per riequilibrare il flusso di lavoro [1].

Quando un Team Scrum applica il metodo Scrumban diventa conveniente suddividere il team in team più piccoli in base alla feature sulla quale devono lavorare (Feature Team), in questo modo quando una feature viene completata gli sviluppatori che se ne occupavano possono essere ricollocati. Il vantaggio dei Feature Team è che riducono il bisogno di produrre artefatti per documentare i risultati di ogni fase dello sviluppo della feature. Inoltre i Feature Team hanno uno scopo preciso e ben delineato, per cui la quantità di lavoro che gli viene asseganta è nota fin dal principio [1].

Il pensiero Lean è maggiormente rivolto allo svolgimento dell'attività adeguata al momento opportuno indipendentemente dal fatto che a compierla sia lo sviluppatore più competente. In questo senso un team Lean dovrebbe rendere la divisione dei ruoli meno rigidi [1].

Non esiste un metodo universale per introdurre un team al metodo Lean e pensare di adottare la filosofia Lean da un giorno all'altro è un'utopia, ma

da qualche parte bisogna cominciare. Il primo passo da compiere è quello di adattare la quantità di lavoro work-in-process alla capacità del team di sviluppo, per farlo bisogna adattare la quantità di attività work-in-process alle proprie risorse. Una volta raggiunto questo primo livello di controllo è possibile cominciare a migliorare il processo di sviluppo.
Migliorare un processo di sviluppo, significa effettuare dei cambiamenti, ma i cambiamenti non sono mai facili da attuare. Le persone non vogliono abbandonare le loro abitudini e fanno resistenza ai cambiamenti che minano la loro posizione. Quindi, i cambiamenti devono affrontare le paure e le resistenze degli individui ed è meglio adottare un approccio evolutivo: piccoli, ma continui cambiamenti [1].

3.2 Il metodo Scrumban

In un sistema di tipo pull, una fase a monte del processo di sviluppo non rilascia i propri requisiti finché gli stadi più a valle non li richiedono. L'uso dei kanban è una tecnica molto diffusa per gestire i trasferimenti di attività tra le varie fasi del flusso di lavoro.

L'idea di usare una semplice Task Board con schede numerate o Post-It è un'idea antica almeno quanto l'introduzione dei modelli Agili. Una Task Board con i tre, ormai classici, stadi: Pending, In Process, Complete; è una variante che ricorda molto la Kanban Board. Nonostante esistano anche software che implementano le Kanban Board, una lavagna fisica è preferibile poiché rispetta due principi Lean: strumenti semplici e visualizzazione del flusso di lavoro. Inoltre, le Kanban Board fisiche sono più facili da modificare e gestire.
La grande novità introdotta dalla Kanban Board è il fatto di aver introdotto dei limiti al numero di Post-It che possono attraversare una stessa fase di sviluppo contemporaneamente. Le iterazioni impongono un limite a quanto

lavoro sia possibile svolgere, ma questa quantità potrebbe essere più elevata rispetto a quella desiderabile. Allo stesso modo un kanban senza limiti è semplicemente una scheda con la descrizione di una feature, non realizza un sistema di tipo pull, né impedisce al sistema di degenerare [1].

Poiché i metodi agili condividono strumenti simili, la loro principale distinzione risiede nella loro filosofia. Diventa, quindi, importante introdurre il Team Scrum alla filosofia Lean adottando gli strumenti tipici di Kanban uno alla volta per dare al team il tempo necessario per comprenderli.

3.2.1 Il processo

Non è difficile aggiungere a Scrum alcune pratiche tipiche dei metodi Lean. La pratica più semplice da integrare è quella di ridurre i tempi delle iterazioni, anche se il processo non è privo di problemi.
L'approccio più semplice è quello di cominciare con delle iterazioni e una pianificazione à la Scrum per poi aggiungere caratteristiche dei sistemi di tipo pull nel processo di sviluppo.

Un modo per introdurre il team alla filosofia Lean è quella di imporre dei limiti al multitasking a cui gli sviluppatori possono essere sottoposti; la regola può anche essere molto semplice: meglio terminare lavori già in stato avanzato di sviluppo piuttosto che cominciarne di nuovi, oppure, si può portare avanti una sola attività alla volta.
Anche se chiunque può farsi carico di più di un'attività alla volta, se tutti gli sviluppatori lavorano su più feature contemporaneamente allora il sistema diventa meno stabile. Il problema è che imporre una regola solo sul singolo sviluppatore ottimizza il problema localmente, ma non globalmente; invece, se si setta anche un limite al numero massimo di feature in lavorazione per tutto il team, si permettono solo alcune eccezioni alla regola generale, questo metodo è più flessibile e, allo stesso tempo, disincentiva comportamenti

controproducenti. A questo punto, si è spostato l'oggetto della regola dallo sviluppatore alle attività, quindi, le schede ora possono essere considerate dei kanban a tutti gli effetti.

Un'altra tecnica che è possibile adottare è quella di assegnare le feature agli sviluppatori all'ultimo momento utile. Infatti, alcuni team pre-assegnano le attività all'inizio dello Sprint; generalmente questa non è una buona idea perché crea dei percorsi critici nel processo di sviluppo. Invece, se si assegnano le feature all'ultimo momento utile si massimizza la conoscenza, poiché tutti gli sviluppatori possono trovarsi a lavorare con qualsiasi parte del software, e si rende il sistema più simile a uno di tipo pull.

Un'altra miglioria che è possibile apportare alla Kanban Board è quello di aggiungere un buffer (Ready) tra il Backlog e le feature In-Progress. La coda Ready contiene attività che appartengono al Backlog, ma che hanno maggiore priorità rispetto alle altre; questi kanban verranno assegnati non appena qualche sviluppatore sarà disponibile per nuove attività. Questa modifica permette di separare la parte di assegnamento delle priorità da quella di assegnazione agli sviluppatori e permette di semplificare quest'ultima fase. Anche la coda Ready deve avere un limite WIP, possibilmente molto basso, visto che il suo scopo è solamente quello di indicare quale sia la prossima feature da implementare.
Dopo tutte queste migliorie si ottiene un sistema di tipo pull in cui rimangono ancora le iterazioni e la pianificazione. Un sistema siffatto può essere considerato Scrumban [1].

Ora che il sistema ha introdotto i concetti di capacità e pull, viene naturale pensare al flusso. Se si spezza lo stadio, molto generico, In-Process in fasi più definite è possibile fornire uno strumento visuale che rappresenta meglio i punti di forza e le debolezze del sistema, in pratica illustra lo stato di salute del processo di sviluppo. Una volta introdotto il concetto di flusso,

viene naturale aggiungere uno strumento per misurarlo: il Cumulative Flow Diagram (CFD). Una semplice Burndown Chart ci informa sul fatto che il team stia o meno rilasciando delle feature, ma non spiega il perché il team stia fallendo o avendo successo nello sviluppo del software. Invece, la CFD comunica molte informazioni riguardo ai tempi di consegna e alla quantità di feature che stazionano nei buffer; queste informazioni sono utili per individuare le fasi problematiche dello sviluppo.
Inoltre, se si definisce il flusso di lavoro in modo più preciso, è possibile sfruttare anche le specializzazioni degli sviluppatori. Infatti, è importante notare che i sistemi di tipo pull non incoraggiano la specializzazione, ma ne traggono vantaggio quando possibile. In fondo è il team che gestisce il flusso di lavoro, quindi è compito del team stabilire quale sia il modo per renderlo più efficiente [1].

Arrivati a questo punto, le fasi di revisione al termine dell'iterazione e la pianificazione avvengono come nel metodo Scrum, ma con un sistema di tipo pull si è reso il flusso di sviluppo più lineare e lo si è migliorato. Inoltre, è possibile avvantaggiarsi dei buffer e del CFD per individuare le opportunità di miglioramento. Mano a mano che si prende dimestichezza con il nuovo metodo, il team si troverà a concentrarsi sempre di più sulla durata delle iterazioni piuttosto che sulla Burndown Chart, poiché la capacità di rilasciare le feature velocemente è la conseguenza e non la causa derivante da un migliore processo di sviluppo [1].

Il tempo medio dei rilasci e la durata dei cicli diventano le prime aree in cui intervenire per migliorare i risultati. Infatti, se il team controlla la durata delle iterazioni e la capacità del team di rilasciare feature è ben bilanciata rispetto alle richieste, allora i tempi dei rilasci sono sotto controllo. Inoltre, se il team controlla la durata delle iterazioni, anche la quantità di feature completate è predicibile, quindi non verranno richiesti sforzi eccessivi e lavoro extra al team di sviluppo. A questo punto lo Sprint Backlog diventa un

semplice "magazzino" con lo scopo di introdurre nuove richieste nel sistema, per cui deve essere minimizzato il più possibile al fine di ottimizzare i costi di pianificazione (pianificare più del dovuto diventa tempo sprecato perché le stesse feature verranno pianificate nuovamente nella successiva iterazione).
Dal punto di vista del team di sviluppo l'utilità dello Sprint Backlog è solo quella di fornire compiti che hanno un alto valore aggiunto. Quindi per controllare la durata delle iterazioni è sufficiente imporre un limite fisso allo Sprint Backlog, invece di valutare quanto tempo richiede ogni feature che viene inserita nel backlog. Infatti, contando il numero medio di feature rilasciate ad ogni iterazione non è più necessario stimare i tempi per le feature nel backlog, ma è sufficiente calcolare la quantità media di feature presenti in esso [1].

Dopo questi ultimi accorgimenti, la pianificazione delle iterazioni avviene ancora a intervalli regolari, insieme alle fasi di revisione e retrospettiva, ma l'obiettivo della pianificazione è quello di aggiungere feature nel backlog in base alla quantità media e non più quello di determinare quante feature è possibile implementare in base alle stime delle singole attività. Questa nuovo approccio riduce enormemente la quantità di tempo dedicata alla pianificazione.

Arrivati a questo punto rimane solo lo scheletro di Scrum, mentre il processo di sviluppo è stato modificato e ottimizzato.

L'uso di modelli Agili porta a implementare feature on-demand, quindi lo Sprint Backlog dovrebbe riflettere lo stato corrente delle richieste. Questo comporta che il backlog sia event-driven e diventa possibile rispondere velocemente alle feature con priorità più alta.
Lo scopo finale è quello di giungere a una pianificazione che introduca nel sistema solo la prossima attività che conviene implementare.
Una pianificazione più estesa non aggiungerebbe alcun valore all'iterazione.

Infatti la pianificazione di Scrum tende a inserire nello Sprint Backlog più lavoro di quanto necessario e questo significa avere delle feature che si accumulano nel "magazzino", cioè uno spreco non necessario.
Per gestire le iterazioni non più su base temporale, ma in base alla necessità è utile stabilire un numero minimo di feature che devono sempre essere presenti nello Sprint Backlog. Quando il numero di attività nel backlog è inferiore al limite, allora diventa utile pianificare quali saranno le prossime feature da inserire nel sistema.
Lo stadio finale di questo processo evolutivo è di giungere a un sistema che sia completamente di tipo pull [1].

Lo stesso ragionamento può essere applicato per ridurre gli intervalli tra i rilasci. Infatti, esiste una quantità ottimale di feature per rilascio, che deve prima essere individuato e successivamente migliorato. Lo scopo ultimo di questo processo è di riuscire a rilasciare feature on-demand.
L'obiettivo di Kanban è quello di massimizzare il risultato del flusso di lavoro dando maggior priorità alle feature che lo richiedono. Regolando la produttività delle fasi che compongono il processo di sviluppo si riesce a raggiungere questo scopo.
In questa missione la sfida più grande è quella di misurare i tempi di consegna, che possono anche essere più lunghi rispetto alla capacità del sistema di effettuare rilasci. Il problema principale si pone nella fase iniziale di adozione del nuovo metodo perché si possono riscontrare intoppi di natura eccezionale e intoppi di tipo ricorrente; inizialmente si dovrà cercare di distinguere tra i due tipi di errori per stimare i tempi dei rilasci in modo realistico. Una volta che si ha un dato da cui partire è possibile modificare il processo di sviluppo per cercare di migliorare i tempi dei rilasci andando a rimuovere le cause dei problemi più gravi e inserendo dei buffer per gestire le problematiche meno urgenti.

Un dato più semplice da controllare è la capacità del flusso. Questo dato

è facile da gestire una volta che il sistema è stabile, invece quando il sistema è appena stato introdotto è necessario inserire la prima feature richiesta non appena possibile, poiché all'inizio più che il problema della gestione del flusso si incontra la difficoltà di crearlo. Il vero problema nella stima della capacità del flusso è che senza un dato certo sui tempi dei rilasci è difficile avere un dato corretto sulla capacità. Questa difficoltà non è di second'ordine perché se la stima è eccessiva il flusso satura e va in stallo, invece, se la stima è troppo bassa il sistema non riesce a essere produttivo al massimo delle sue possibilità.
Quindi, fintanto che non si hanno dati riguardo al tempo medio dei rilasci è necessario aggiustare la capacità del flusso di sviluppo con tentativi empirici; in questi casi è possibile avvalersi della Kanban Board per individuare quale sia l'andamento del flusso e intervenire di conseguenza.
Un buon metodo per adottare Scrumban e ottenere tutti i dati necessari senza rischiare di mandare in stallo l'intero sistema è quello di usare un team pilota che sperimenta gli strumenti Scrumban. Una volta che il flusso di lavoro del team pilota si è stabilizzato e si è riusciti a ottenere dati più concreti riguardo ai tempi dei rilasci è possibile estendere Scrumban agli altri team dell'organizzazione [1].

Il Daily Meeting è una pratica comune a tutti i metodi Agili. In particolare, il Daily Standup Meeting è diventato un riferimento dal punto di vista culturale nel mondo degli sviluppatori.
Solitamente lo Standup Meeting ha lo scopo di fortificare le relazioni tra i membri del team. Nel Daily Scrum gli sviluppatori enunciano brevemente cosa stanno facendo per il progetto. L'idea dietro questo tipo di meeting è che la comunicazione all'interno di progetti complessi diventi talmente costosa da limitare la produttività dei partecipanti. Questi meeting garantiscono una comunicazione tra tutti gli sviluppatori almeno una volta al giorno. Inoltre, il costo di questa comunicazione è limitato dal fatto che il meeting ha una durata di 15 minuti; se sono richiesti ulteriori approfondimenti o chiarimenti

i membri del team che sono interessati all'argomento possono continuare il meeting tra di loro. Anche il fatto che il meeting abbia un tempo fissato pone un limite al numero di partecipanti, che definisce anche la grandezza massima di un team Scrum. Un progetto che richieda più sviluppatori deve usare lo Scrum di Scrum.
Invece, Kanban riesce a scalare senza troppi problemi fino a team di 40 sviluppatori. Solitamente un team Kanban si considera di grandezza ottimale quando è composto da 25 membri. Oltre questa dimensione, probabilmente, conviene dividere il team in sotto-team che lavorino parallelamente; è importante, però, tenere conto anche dei maggiori costi generati dalla gestione di più team, quindi è meglio rimandare la scorporazione di un team finché non si presentano i primi segnali di inefficienze [1].

La particolarità dei Kanban Standup è che il direttore del meeting comunica le attività da svolgere e non di chi ne verrà incaricato, poiché l'aspettativa è che ogni stadio rappresentato sulla Kanban Board accoglierà una nuova feature appena avrà capacità disponibile; a questo punto qualsiasi sviluppatore può farsi carico di questa attività e lo fa in modo autonomo senza aspettare che il lavoro gli venga assegnato. Se il direttore del meeting nota che ci sono dei compiti non assegnati, allora cerca di spingere il team a farsene carico. Inoltre, deve cercare di motivare e guidare il team. Anche il team deve partecipare alla pianificazione individuando opportunità di miglioramento del processo di sviluppo. L'assunzione da cui parte il meeting è che il processo di sviluppo sia corretto e che il flusso scorra. Le discussioni devono essere limitate ai casi eccezionali: se le feature riescono a essere rilasciate il sistema può essere considerato funzionante, invece se uno sviluppatore si trova bloccato, allora viene aperta una discussione su come risolvere il problema. Quando si fa una panoramica delle attività sulla Kanban Board è utile analizzare la situazione andando da destra a sinistra allo scopo di enfatizzare la visualizzazione del meccanismo pull.

La pianificazione di un'iterazione è un altro elemento fondante di Scrum.

L'obiettivo di Scrum è di salvaguardare il lavoro del team di sviluppo dalle interferenze esterne, quindi un'iterazione di quattro settimane include una parte di pianificazione dove i clienti possono chiedere aggiustamenti. Una volta terminata questa fase si è stabilità una priorità tra le feature e il team stima i tempi necessari per svilupparle, poi sceglie quali requisiti devono essere sviluppati nella successiva iterazione. La ratio è che esiste un momento in cui il cliente può intervenire nel progetto, per il resto del tempo dovrà aspettare che il team abbia completato ciò che era stato richiesto. Questo approccio potrebbe portare a non terminare mai lo sviluppo del software se il cliente continua a cambiare idea, ma il vero obiettivo è quello di rilasciare servizi di buona qualità. La cosa importante è che le attività siano descritte in modo abbastanza chiaro e che siano stimate correttamente; se questo avviene, è sufficiente che ogni feature non richieda più tempo di quanto ne sia disponibile in un'iterazione. Anche se le feature hanno complessità diverse e quindi tempi di realizzazione differenti, in media il numero di feature rilasciate ad ogni iterazione è costante. Di conseguenza è possibile usare questo dato per fare rilasci anche senza l'uso delle iterazioni. La fase di pianificazione deve, però, essere abbastanza frequente da evitare lo stallo per mancanza di attività disponibili. Invece, l'intervallo tra un rilascio e l'altro deve trovare un equilibrio tra il costo di un nuovo rilascio e il costo di tenere delle feature complete in "magazzino". Quindi, se i costi dei rilasci sono alti, si faranno pochi rilasci; invece, se i costi dei rilasci sono bassi è possibile pensare anche dei rilasci on-demand [1].

L'obiettivo del processo di sviluppo è quello di rilasciare quante più feature possibile rispetto alle richieste del cliente. Le due variabili che influiscono maggiormente sul throughput del sistema sono la quantità di feature in fase di sviluppo e la durata delle iterazioni. La gestione delle feature in sviluppo viene controllata dal team usando i limiti WIP. Invece, la durata delle iterazioni è più complicata da gestire perché molti dei ritardi che si verificano nel processo di sviluppo si palesano nel mezzo di una iterazione. Queste

variazioni possono essere controllate usando alcuni accorgimenti:

- Suddividere le feature troppo grandi in attività di dimensione più piccola.
- Usare pratiche di programmazione (come i design pattern) per evitare di dover effettuare refactoring del codice in futuro.
- Non perdere mai di vista i problemi che causano lo stallo del processo e intervenire per risolverli.

La durata delle iterazioni non è importante per la motivazione del team, ma è rilevante solo come dato da tenere sotto controllo. Infatti, come primo passo è necessario rendere stabile il sistema e in seconda fase modificare il sistema per cercare di migliorarlo (questa fase può ovviamente portare a instabilità) per poi ritornare alla fase di stabilizzazione del sistema; la stabilità del sistema è un fattore essenziale per poterlo migliorare ed è, quindi, sempre il primo obiettivo da raggiungere. Inoltre, se il sistema non riesce mai a rispettare le aspettative allora il problema non è il sistema ma le aspettative [1].

La risoluzione dei bug è importante perché le feature che rimangono bloccate sono valide per il conteggio del limite WIP. Quando si ha un kanban bloccato si possono scegliere due strade:

1. reinserire il kanban in uno degli stadi iniziali
2. assegnare i bug a uno stadio predefinito per questo scopo nella Kanban Board

Il primo approccio è quello più semplice, ma che dà minore priorità alla risoluzione dei bug. Una controindicazione di questo metodo è che tratta la risoluzione dei bug come una qualsiasi altra attività, quindi se il team è già impegnato in altri lavori la risoluzione del problema rimarrà in attesa. L'idea alla base di questo approccio è che i bug non sono inaspettati.

Il secondo metodo è più complesso da gestire poiché tratta i bug come un fallimento del processo di sviluppo che devono essere affrontati il prima possibile. Il fatto di avere uno stadio del flusso di sviluppo dedicato alla risoluzione dei bug permette di intervenire immediatamente. La controindicazione è che la capacità del team di effettuare rilasci viene ridotta perché parte di questa forza lavoro viene dedicata alla risoluzione di problemi invece che alla creazione di nuovo valore aggiunto. Questo approccio considera i bug come situazioni eccezionali che richiedono un intervento correttivo rapido e risulta utile per i team che stanno lavorando su un nuovo progetto o se viene richiesto un livello qualitativo molto alto del software.

L'obiettivo finale è quello di ridurre i tempi di rilascio delle richieste del cliente, questo obiettivo viene raggiunto controllando attentamente quante feature rimangono in attesa nel processo di sviluppo.
Un problema che sembra presentarsi regolarmente per i team è quello di come suddividere le attività tra gli sviluppatori. Un approccio è quello di suddividere i compiti in base alle aree di competenza. Un altro, invece, è quello di assegnare delle priorità alle feature e assegnarle a tutto il team. Un sistema che faccia uso di Kanban permette di usare una soluzione mista: si dividono le attività in aree di competenza senza assegnarle a nessun membro del team, sarà poi uno sviluppatore specializzato a farsi carico di una feature che appartiene alla sua area di competenza [1].

I tempi dei rilasci dipendono sia dal throughput che dalla quantità di feature in stato di sviluppo. Infatti, dato un tasso di throughput (considerando impegnati tutti gli sviluppatori), un incremento dei limiti WIP comporta un allungamento dei tempi dei rilasci, poiché un innalzamento dei limiti WIP significa più lavoro, quindi un maggior sforzo da parte del team; se gli sviluppatori sono già impegnati al massimo delle loro capacità servirà più tempo per svolgere il carico di lavoro aggiuntivo.
Una tecnica molto semplice che il management può adottare è quella di sem-

plificare il problema usando le Policy. Poiché i tempi di consegna dipendono dai limiti WIP, è possibile imporre di mantenere fissi questi limiti concedendo solo piccolissime variazioni; successivamente è possibile intervenire sul throughput. I limiti WIP sono facili da gestire perché un membro dell'organizzazione decide se accettare o rifiutare una nuova richiesta del cliente e ridurre la quantità di lavoro in entrata nel sistema è sicuramente più facile che migliorare il processo di sviluppo. Quando si ha sotto controllo una variabile diventa più semplice intervenire sull'altra.

Se il team ha un flusso di lavoro ben ottimizzato, allora il numero complessivo di feature in sviluppo è pari alla somma dei limiti WIP di tutte le fasi del processo di sviluppo. Va notato che un limite WIP globale non impedisce variazioni nella distribuzione delle feature nei vari stadi del sistema. Quindi, dopo aver imposto un limite WIP globale è possibile gestire in modo più stringente i limiti WIP locali, il risultato è che alcune fasi mostreranno una maggior correlazione con i tempi di consegna rispetto ad altre.

Il fatto che le feature entrino nel sistema in modo saltuario e la natura creativa e intellettuale dell'attività di costruzione di un software portano spesso il flusso di sviluppo a rompere i meccanismi di sincronizzazione. Il rischio e l'incertezza sono insiti nella natura dello sviluppo software, ma la novità è ciò che fornisce valore aggiunto al prodotto, quindi è possibile ridurre i rischi dovuti ad imprevisti per un certo periodo oltre il quale è possibile solo mitigarli e accettarli come effetti collaterali dell'attività di sviluppo [1]. Il team può usare piccoli buffer come intermezzo tra i vari stadi del processo di sviluppo per assorbire le variazioni nei tempi di sviluppo delle feature. I kanban possono attraversare i buffer allo stesso modo in cui attraversano le altre colonne della Kanban Board; anche questi kanban sono validi per il conteggio totale.

In condizioni di sviluppo normali, le code Kanban dovrebbero operare al

di sotto del loro limite; quindi, il sistema è lasco. Il fatto che il sistema sia allentato è un aspetto positivo, infatti un sistema ottimizzato è un sistema sufficientemente lasco. Il metodo con cui si assegnano i limiti non è uguale per tutte le code e i buffer, che sono code che non generano valore aggiunto, devono avere dei limiti che siano i più bassi possibili. Le code hanno lo scopo di agevolare il flusso, quindi se sono troppo grandi tenderanno ad accumulare troppe attività in un'unico stadio, invece, se sono troppo piccole possono bloccare il flusso. La grandezza giusta per una coda è "il limite più basso possibile per cui il sistema non va in stallo il X% delle volte". La differenza tra 100 e X corrisponde alle aspettative di miglioramento del processo [1].
La salute del sistema è data dalla salute delle sue singole parti, infatti ogni coda costituisce un collo di bottiglia per la coda precedente e se una di queste si satura può mandare in blocco il flusso di tutto il sistema.

In conclusione, una volta che il processo di sviluppo viene gestito come uno flusso e tutti gli strumenti Kanban vengono usati dal team, quest'ultimo è in grado di abbandonare il metodo Scrum e adottare Kanban.

3.2.2 I ruoli

Anche per quanto riguarda i ruoli dei membri del team è possibile effettuare una transizione da Scrum a Kanban. Partendo dagli incarichi già presenti in Scrum si può effettuare una transizione verso Kanban riassegnando le mansioni, cercando il più possibile di mantenere una linearità delle responsabilità nel passaggio.

Per quanto riguarda il Development Team non ci sono modifiche da effettuare, anche se Kanban permette di integrare più Team Scrum in un team di dimensioni maggiori. Questa scelta può essere conveniente per ridurre i costi di di gestione.

Il Product Owner può diventare il Service Request Manager nella nuova conformazione del team, infatti la sua capacità di comunicare con il cliente e di comprendere quale sia la priorità da assegnare a ogni feature sono qualità utili in entrambi i ruoli.

Lo Scrum Master è la figura probabilmente più indicata per prendere il ruolo di Service Delivery Manager. Infatti la sua abilità di gestire uno Scrum Team può essere sfruttata anche nel nuovo organigramma per indirizzare le attività del team.

Il ruolo del Kanban Coach può richiedere l'introduzione di una nuova persona all'interno dell'organizzazione se nessun membro del team ha già avuto esperienze nel mondo Lean o Kanban. Con il passare del tempo, quando il team ha preso dimestichezza con il nuovo metodo, questo ruolo può essere assegnato al Service Delivery Manager.

3.2.3 Le metriche di Scrumban

Come già evidenziato per i ruoli, anche per le metriche Scrumban può effettuare una transizione da quelle maggiormente legate a Scrum a quelle introdotte da Kanban. Sarà compito del Kanban Coach quello di introdurre le metriche Kanban e abbandonare quelle Scrum in base alle varie fasi del processo di transizione da un metodo all'altro.

Capitolo 4

Kanban Board virtuali: Taiga

4.1 Cos'è Taiga

Negli ultimi anni i metodi Agili hanno conosciuto un enorme successo, soprattutto nell'ambito delle Startup. La diffusione di nuovi metodi gestionali ha richiesto la creazione di nuovi strumenti per organizzare un team agile. Molti di questi sono stati pensati per team che adottano Scrum o Kanban e vengono gestiti come servizi cloud per permettere ai membri del team di accedervi in qualsiasi momento mantenendo sincronizzate le informazioni tra i dispositivi e gli utenti coinvolti.

Taiga è un software per il project management basato sul cloud sviluppato dal team Taiga Agile, rilasciato per la prima volta nel 2014. Il codice di questo prodotto è fruibile tramite Github sotto licenza AGPL (Affero General Public License), per cui può essere annoverato tra i software opensource. È possibile accedere alla piattaforma tramite web browser all'indirizzo `https://taiga.io/` o tramite app per i dispositivi supportati (è presente sugli store Apple, Android e Windows).
Per facilitare la transizione di progetti già in sviluppo verso questa piattaforma è possibile importarli da altri servizi come Trello, Jira, Asana e Github.

4.2 Funzionalità di Taiga

Taiga permette di creare due tipi differenti di progetti: Scrum e Kanban. Al momento della creazione di un nuovo progetto, l'utente ha la facoltà di scegliere tra i due template disponibili. Una volta istanziato il progetto è possibile aggiungere altri utenti come membri del team.
Esiste anche la possibilità di aggiungere dei template personalizzati duplicando un progetto esistente e modificandone i parametri, i moduli e i ruoli a piacere; quando la configurazione del nuovo progetto è completa diventa possibile duplicarlo nuovamente e sfruttarlo come template per altri progetti.

Il template del progetto può essere modificato con l'aggiunta o la rimozione di moduli: Epics, Backlog, Kanban, Issues, Wiki, Meet Up. Per i progetti Scrum sono attivati: Backlog, Issues e Wiki; mentre per i progetti Kanban è attiva solo la Kanban Board. Attivando sia il Backlog che la Kanban Board è possibile ottenere un template valido per Scrumban.
Tra i moduli disponibili l'unico che prevede integrazioni con altri software è Meet Up, utile per integrare dei sistemi di videoconferenze all'interno di Taiga poiché questa piattaforma è sprovvista di sistemi di comunicazione real-time.

Inoltre è possibile integrare Taiga con Slack e Hip Chat per ricevere notifiche riguardo a Epics, User Stories, Tasks, Issues e Wiki, con la possibilità di scegliere per quale tipo di attività (creazione, modifica, eliminazione) si vuole ricevere una notifica.

4.2.1 I ruoli

I ruoli previsti per i membri di un progetto sono gli stessi per entrambi i metodi: Product Owner, UX, Design, Front, Back, Stakeholder, External User. Al creatore del progetto viene assegnato di default il ruolo di Product Owner. Per poter assegnare un ruolo a un membro del team è necessario che sia un utente Taiga, fa eccezione il ruolo di External User che può essere

assegnato anche a utenti non iscritti alla piattaforma. Inoltre, è possibile creare nuovi ruoli e rimuovere quelli presenti di default.

I ruoli, hanno non solo lo scopo di distinguere gli ambiti di intervento dei vari membri del team, ma servono anche per gestire i permessi che si vogliono assegnare a ogni tipologia di ruolo. Le categorie di permessi sono: Epics, Sprints, User Stories, Tasks, Issues, Wiki; per ognuna di esse è possibile assegnare i permessi in 5 livelli: sola lettura, possibilità di aggiungere un nuovo item, possibilità di modificare un item, possibilità di commentare un item, cancellare un item.

4.2.2 Scrum Project

Un progetto Scrum prevede un Backlog nel quale inserire tutte le User Stories individuate. Per ognuna di esse è possibile assegnare lo sforzo richiesto con un sistema a punti il cui significato deve essere stabilito dal team; una caratteristica interessante di questa feature è che ogni team funzionale può assegnare un proprio punteggio ad ogni User Story e lo sforzo totale che essa richiede è dato dalla somma dei punteggi assegnati da ogni team funzionale.

Le User Stories presenti nel backlog possono essere assegnate al prossimo Sprint, in questo caso vengono inserite nella Sprint Taskboard. Quest'ultima è strutturata nel modo seguente: nella colonna sinistra si trovano le User Stories, mentre le altre colonne sono dedicate ai task relativi alle User Stories, che vengono creati in questa fase per avere un controllo più raffinato sullo sviluppo. La rappresentazione della Sprint Taskboard rispecchia la stessa struttura presentata nel capitolo su Scrum, quella fornita è composta dalle seguenti colonne: New, In Progress, Ready For Test, Closed, Needs Info. Le User Stories possono essere assegnate a uno o più sviluppatori e lo stesso vale per i singoli task che le compongono. Modificando gli stati assegnabili ai task è possibile modificare la struttura della Sprint Taskboard.

Insieme alla Sprint Taskboard è possibile visualizzare un grafico equivalente allo Sprint Burndown Chart, mentre nel Backlog è presente una visualizzazione della Release Burndown Chart .
Le User Stories e i task possono anche essere contrassegnati come "bloccati" con la possibilità di segnalare il motivo di questa decisione.

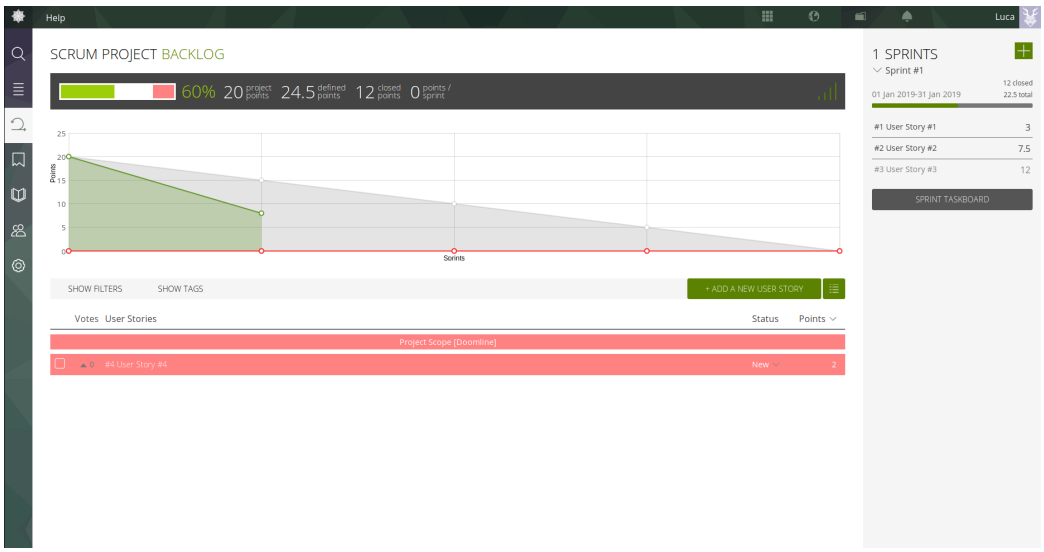

Figura 4.1: Scrum Backlog in un progetto Taiga

Nella sezione Issues è possibile inserire i problemi e i bug riscontrati con la possibilità di assegnarli a uno o più membri del team e di evidenziarne la gravità e lo stato di avanzamento.

Nella sezione Wiki si ha la possibilità di creare una documentazione composta formattando il testo con Markdown. La struttura può essere arbitrariamente complessa poiché è possibile suddividere la documentazione in più pagine.

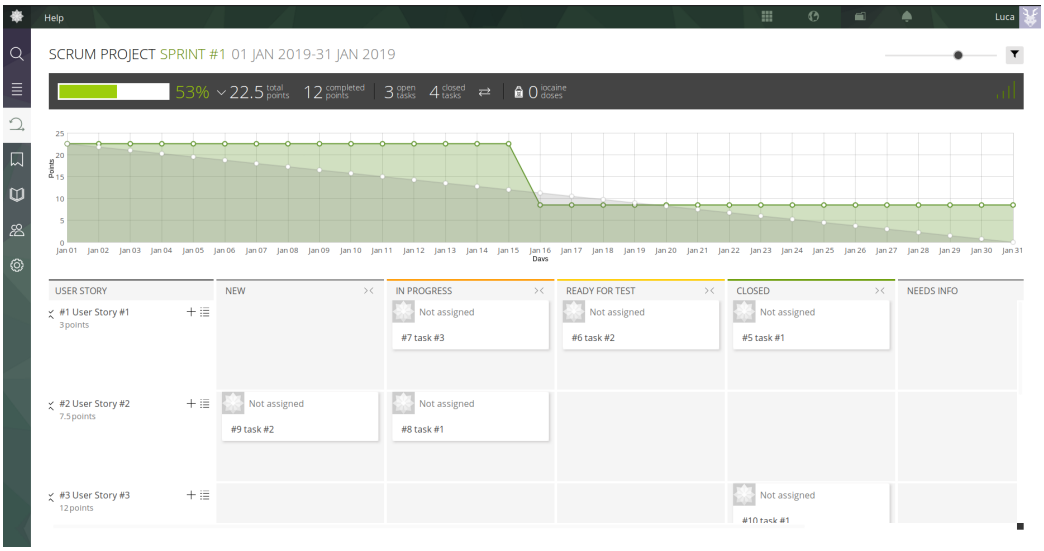

Figura 4.2: Scrum Taskboard in un progetto Taiga

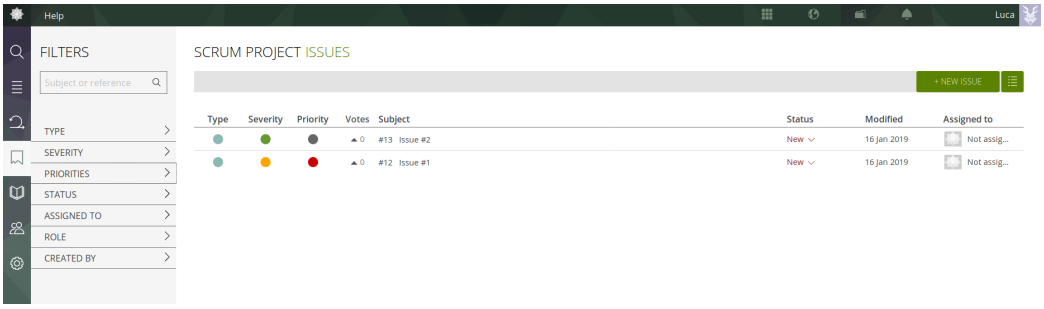

Figura 4.3: Issues in un progetto Taiga

4.2.3 Kanban Project

Un progetto Kanban prevede solamente la Kanban Board. La struttura fornita contiene le colonne New, Ready, In Progress, Ready For Test, Done, Archived, ma la struttura è adattabile dal pannello di controllo modificando gli stati possibili per una User Story. Inoltre, è possibile assegnare un limite WIP al numero di User Stories per ogni fase del processo di sviluppo.
Nella Kanban Board non è possibile creare task per le User Stories poiché quest'ultime vengono considerate unità atomiche, come previsto dal metodo Kanban.

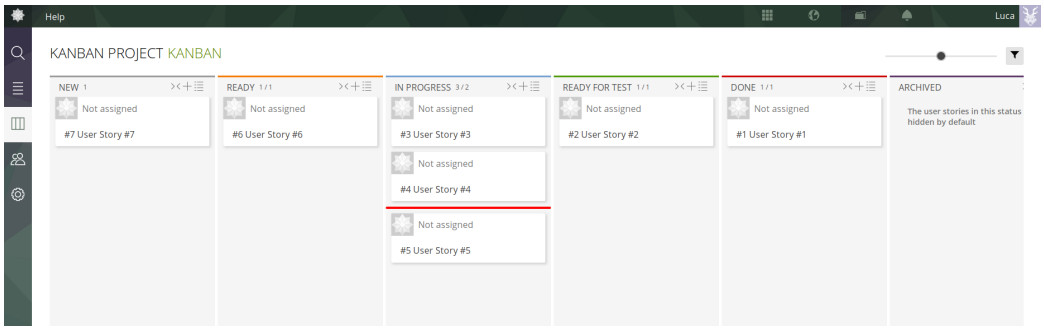

Figura 4.4: Kanban Board in un progetto Taiga

Mentre l'impostazione di default prevede solamente la Kanban Board, è possibile aggiungere il modulo Epics per avere una gestione visiva delle release. Infatti questo modulo permette di creare una o più release e di assegnare a ognuna di esse delle User Stories, in questo modo è possibile visualizzare in una barra di avanzamento lo stato della release. Questo modulo può, ovviamente, risultare utile anche in un progetto Scrum, anche se lo Sprint Backlog fornisce già qualche informazione utile riguardo al prossimo rilascio.
Il modulo Epics risulta conveniente quando si vuole organizzare il lavoro su più release, per questo motivo non è stato introdotto nel template Kanban: per non contravvenire alla regola di prendere decisioni all'ultimo momento utile.

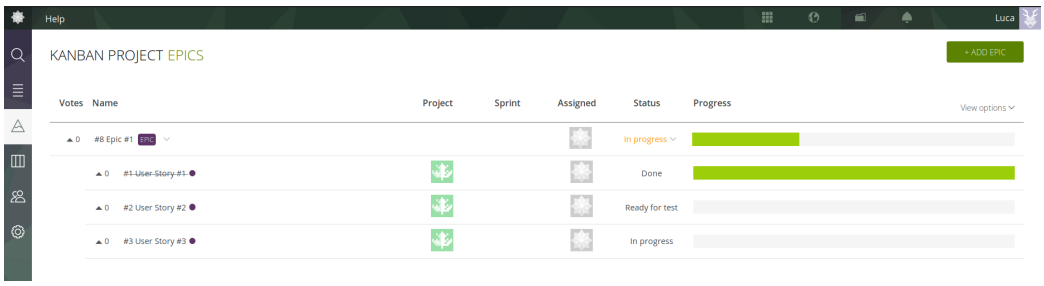

Figura 4.5: Epics in un progetto Taiga

4.3 Creare un Template per Scrumban

La creazione di un template adatto per Scrumban non è semplice poiché questo metodo non è "stabile" nel tempo, ma in continua evoluzione.

In questo senso Scrumban costituisce una fase di transizione da Scrum a Kanban, per cui sembra ragionevole partire dal template per Scrum per poi evolvere verso il template per Kanban. Questo significa partire da un progetto che contenga i moduli: Backlog, Issues e Wiki. Un primo passo che il team può compiere per approcciarsi a Scrumban è quello di non usare la Task Board presente nel modulo Backlog, ma di aggiungere il modulo Kanban Board e usarlo per la gestione delle User Stories. In seguito, il team potrà decidere se rimuovere i moduli Issues e Wiki per raggiungere un approccio Lean completo o se vorrà mantenere questi moduli per continuare a usare un metodo ibrido.

Invece, se si vuole adottare Scrumban come metodo "stabile" è possibile creare un template adatto inserendo solamente i moduli Backlog e Kanban Board.
In realtà, è possibile considerare anche l'introduzione del modulo Issues. Infatti, nel capitolo su Scrumban sono stati analizzati due metodi differenti per la gestione dei bug: il primo prevede di reinserire i kanban nella prima colonna della Kanban Board per ripercorrere il processo di sviluppo, invece il secondo approccio prevede di inserire i kanban in un apposito stadio della Kanban Board. Se un team adotta il primo metodo allora non necessita del modulo Issues, invece se predilige il secondo allora può usufruire di questo modulo per evitare di aggiungere un'ulteriore colonna alla Kanban Board.

Per quanto riguarda la gestione dei ruoli non ci sono modifiche rilevanti da effettuare. Infatti la divisione degli sviluppatori in team funzionali risulta comoda anche in questo metodo, mentre il Product Owner e lo Scrum Master possono essere rinominati rispettivamente in Service Request Manager e

Service Delivery Manager, ma questo aspetto è una pura formalità che non cambia il modo in cui il team interpreta i ruoli assegnati. L'unico aspetto rilevante è quello dei permessi assegnati a questi due ruoli, anche se, in generale, non necessitano di cambiamenti.

4.4 Considerazioni finali

In generale, Taiga è un software altamente personalizzabile con un pannello di controllo ricco di funzionalità. L'utente è in grado di adattare il processo di sviluppo alle proprie esigenze grazie ai moduli disponibili e di riprodurre l'organizzazione del team all'interno del software in modo realistico modificando i ruoli a disposizione e i relativi permessi.
Inoltre, è possibile personalizzare il software anche a livello di semplice gusto estetico poiché i colori assegnati agli stati dei vari item sono tutti modificabili a piacere.

Taiga fornisce anche la possibilità di generare dei report su Epics, User Stories, Tasks e Issues in formato CSV, direttamente scaricabili attraverso un comodo link generato automaticamente dalla piattaforma. Viene fornita anche la possibilità di esportare un progetto in un file formato JSON che contiene tutte le informazioni riguardo al progetto: si parte dalle informazioni generali, ma comprende anche i ruoli e i loro permessi, gli item con i relativi status, le impostazioni specifiche del progetto; in generale, qualsiasi parametro modificabile viene inserito all'interno del report, che può essere usato per esportare il progetto su un nuovo utente o semplicemente per avere un backup disponibile.

4.5 Taiga vs Trello

Taiga non è l'unico software per il project management in circolazione. Tra i suoi principali concorrenti spicca Trello, anch'esso un software per il project management basato su cloud rilasciato per la prima volta nel 2011. Al contrario di Taiga, Trello è un software proprietario detenuto da Atlassian Corporation. È possibile accedere a Trello tramite browser web all'indirizzo `https://trello.com` oppure con le applicazioni ufficiali presenti negli store Apple, Android e Windows.

Trello si propone all'utente in modo più semplice rispetto a Taiga, l'unico strumento che viene fornito è la Board. Non sono previsti template ufficiali come per Taiga, ma è possibile sfruttare anche in questo caso la possibilità di copiare delle Board per usufruire di una struttura già organizzata. Trello fornisce dei template creati dagli utenti e approvati dal team Trello all'indirizzo `https://trello.com/inspiration`, dove è possibile reperire template per gli scopi più vari (anche non relativi alla programmazione, come per esempio l'organizzazione di un matrimonio).

Trello non fornisce nemmeno moduli aggiuntivi per il backlog o la gestione dei bug, ma spetta all'utente organizzare la Board in modo da gestire tutti gli aspetti del management nel modo che preferisce. Una soluzione molto diffusa è quella di creare una colonna iniziale intitolata Backlog con uno scopo ben palesato dal suo nome.

Un campo in cui Trello fornisce molta più flessibilità rispetto a Taiga è quello delle integrazioni con altri software. Trello comprende delle "Add-On" per una gran quantità di servizi ed è quindi in grado di integrare software per la comunicazione real-time, calendari, servizi per lo storage di file nel cloud, sistemi di votazione e tante altre funzionalità aggiuntive.

Invece, per quanto riguarda i ruoli Trello non fornisce lo stesso grado di ac-

curatezza che viene reso disponibile da Taiga. Trello non prevede la creazione di ruoli, quindi sono disponibili solo due tipologie di utente (relativamente a una Board): amministratore e membro del team. Gli amministratori hanno, ovviamente, un controllo completo sulla Board con la possibilità di modificarne le colonne e il contenuto, ma anche di aggiungere e rimuovere utenti dal team. Invece, il membro del team può modificare solo le Cards presenti nella Baord, senza la possibilità di modificarne la struttura.
Trello fornisce anche la possiblità per i suoi utenti di "seguire" una Board e, se le impostazioni della Board lo permettono, di lasciare dei commenti. Questa funzionalità costituisce l'equivalente di quello che in Taiga è l'External User.

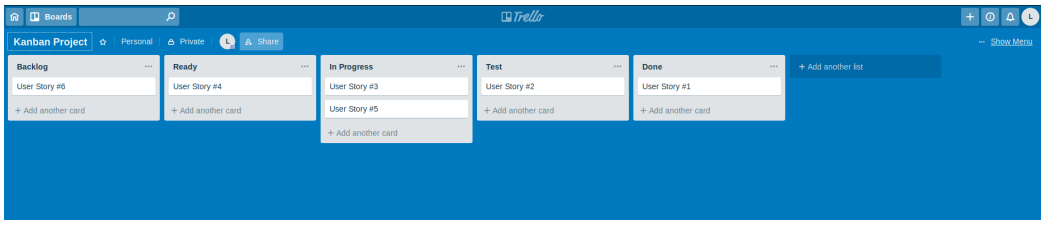

Figura 4.6: Esempio di una Board in Trello

Infine, Trello permette di esportare le Board in formato CSV o JSON, anche se non fornisce alcun metodo per importare progetti generati da altri software o dal backup di una Board; infatti la soluzione proposta sul sito ufficiale consiglia l'importazione di un file CSV tramite l'utilizzo di un software esterno "Import2 Wizard" (`https://help.trello.com/article/751-importing-data-into-trello`).

In definitiva, Trello non costituisce uno strumento per il project management raffinato come Taiga per diversi motivi:

- il fatto che molte funzionalità siano delegate a servizi esterni può costituire un problema a livello di omogeneità nei permessi assegnati;

- i ruoli hanno dei permessi prestabiliti e non modificabili, non è quindi possibile riprodurre sul piano virtuale quella che è l'effettiva organizzazione del team nel mondo reale;

- le Board di Trello non sono Kanban Board poiché sono sprovviste del limite WIP.

Trello è sicuramente un software semplice ed efficace, che non richiede conoscenze pregresse; inoltre, la gran quantità di integrazioni con altri servizi lo rende adattabile alle situazioni più disparate. Dunque si presta molto per essere usato per progetti amatoriali o per progetti già esistenti per cui sarebbe scomodo introdurre un software per il management che sia troppo complesso o che richieda di modificare alcune pratiche consolidate per il team.
Invece Taiga si presenta come un prodotto più maturo (come livello concettuale), che richiede ai membri del team di condividere una struttura organizzativa complessa. Sicuramente è un software meno adattabile rispetto a Trello, ma è molto flessibile nella sua struttura e fornisce gran parte delle funzionalità indispensabili a un team per collabare, mentre per le funzionalità che non riesce a fornire rende disponibili delle integrazioni con software che le implementano.

In base alla situazione è preferibile usare un software piuttosto che l'altro, ma è la mentalità del team e la dimestichezza che i suoi membri hanno con i concetti dell'ingegneria del software che rendono un tool preferibile all'altro.

4.6 Taiga vs UpWave

Taiga e Trello sono due software per il project management che hanno riscontrato un forte successo negli ultimi anni, ma esiste un altro concorrente che sta prendendo piede: UpWave.
Al contrario dei primi due, UpWave è un software privato che può essere

usato esclusivamente da chi ne ha acquistato la licenza, anche se fornisce 14 giorni di prova gratuita. Il suo mercato di riferimento è, dunque, quello delle aziende software.

Una prima differenza che è possibile riscontrare rispetto a Taiga e Trello è che UpWave non rende disponibili app per nessun tipo di dispositivo o sistema operativo, l'unico modo per accedervi è tramite il sito ufficiale `https://www.upwave.io/`.

Una volta effettuato l'accesso è necessario costituire almeno un team, questo passaggio è fondamentale perché ogni Board deve essere assegnata a un team. La costruzione del team è simile a quella di Trello: non sono previsti altri ruoli se non quello di amministratore della Board e quello di semplice utente.

Terminata la creazione del team è possibile iniziare la fase di creazione della Board, che consiste nello scegliere un nome per la Board (che solitamente corrisponde al nome del progetto) e un team a cui assegnarla; prima di completare la creazione della Board è possibile scegliere un template da cui partire. In questo senso UpWave è più simile a Trello che a Taiga: i template disponibili sono numerosi e spaziano su modelli e argomenti di vario genere. I template sono quelli inclusi in UpWave, ma è possibile crearne di propri, non esiste invece la possibilità di cercare template sviluppati da altri utenti.

Le Board di UpWave si distinguono da quelle di Trello per essere delle vere e proprie TaskBoard di Scrum, mentre quelle di Trello sono più simili a delle Kanban Board. Infatti le Board di UpWave sono già strutturate per ospitare le User Stories e i task possono essere inseriti non solo nella fase corretta (colonna), ma anche nel componente adeguato (riga).
Esiste anche la possibilità di trasformare le Board da Task Board a Kanban Board, in questo modo le Board di UpWave diventano identiche a quelle di

Trello.

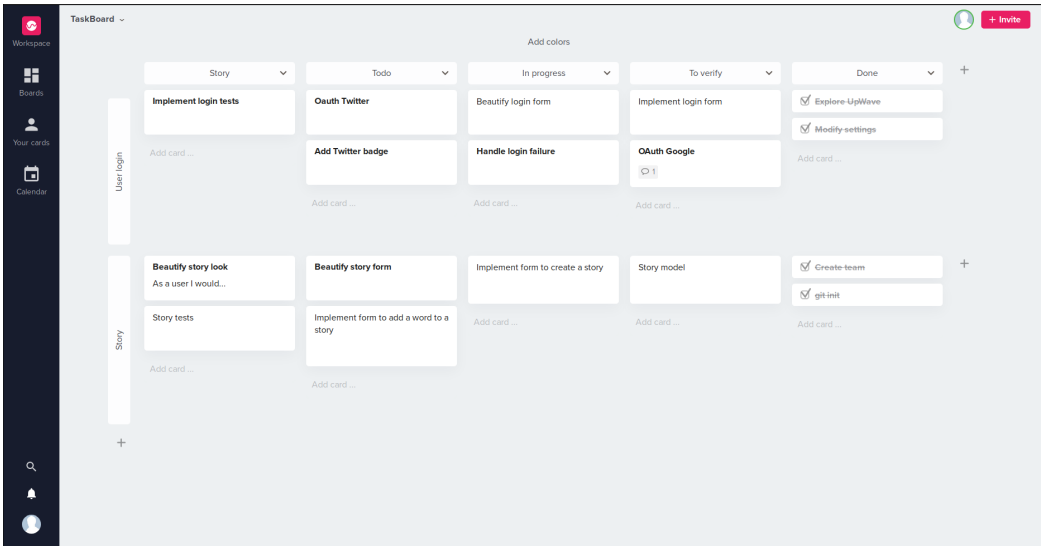

Figura 4.7: Esempio di Board in UpWave

La gestione delle Board in UpWave, per ora, non si distingue molto da quella di Trello o Taiga, ci sono, però, delle feature che vengono proposte in questo software e che non sono disponibili negli altri due.
Una prima feature molto interessante per le aziende è il time tracking: per ogni task è possibile registrare il tempo che è stato impiegato per completarlo; è anche possibile fornire delle stime per ogni task, in questo modo è facile verificare se le stime sono efficaci o da migliorare.
Inoltre le Board possono essere collegate a Slack, a Google Calendar e Zapier. Questi plug-in possono risultare utili per un team poiché integrano all'interno di un unico software meccanismi per la gestione dello sviluppo del progetto (UpWave), della comunicazione (Slack) e dell'organizzazione del team (GCal) e, infine, per automatizzare alcune operazioni (Zapier).
Invece la gestione dei task non aggiunge feature rispetto a quelle che sono già state riscontrate in Taiga e Trello, come l'assegnamento del task a un membro del team, la possibilità di creare dei sub-task, impostare una scadenza e

altre piccole personalizzazioni (come quella di assegnare dei colori ai task).

L'aspetto che distingue maggiormente UpWave da Taiga e Trello è la metrica. Per ogni Board è possibile ottenere un report, che mostra un grafico simile allo Sprint Burndown Chart (anche se in UpWave non esiste il concetto di Sprint, infatti il grafico mostra l'andamento dei task negli ultimi 30 giorni) e i dati riguardanti il numero di task completati e i relativi tempi.
Oltre al report sui task è possibile ottenere un "Time Sheet", un grafico che illustra il tempo speso dal team sul progetto suddividendolo in base agli utenti o ai task.
Inoltre, ogni utente ha pagina Analytics dedicata in cui viene esposto un sommario delle proprie attività negli ultimi 30 giorni tramite grafici:

- Workspace Activity: mostra le azioni effettuate sulle Board a cui partecipa l'utente.
- Created vs Completed: un grafico a torta che illustra il rapporto tra task attivi e task completati creati dall'utente.
- Most Active Boards: un grafico a torta che mostra il rapporto tra i tempi dedicati dall'utente a ogni Board.
- Cards Completed: un grafico che illustra l'andamento dei task completati dall'utente.
- Comments: un grafico che mostra il numero di commenti che l'utente ha scritto nelle card.

Infine, UpWave permette all'utente di esportare le Board in formato JSON o CSV; questa feature è comune a tutti e tre i software che sono stati presentati.

In generale, UpWave si presenta come un buon software per la gestione di un progetto, le integrazioni con software di uso comune lo rendono molto comodo da integrare in un ambiente già operativo, ma anche facilmente

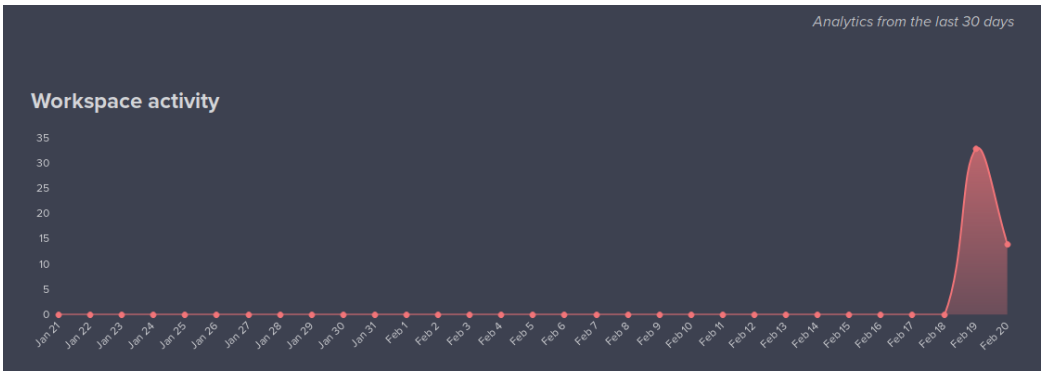

Figura 4.8: Grafico "Workspace activity" in UpWave

Figura 4.9: Grafici "Created vs Completed" e "Most active boards" in UpWave

configurabile per nuove realtà aziendali. Il fatto che sia raggiungibile solo tramite interfaccia web e che il codice sorgente sia chiuso lo rende meno flessibile come strumento rispetto a Taiga e meno conveniente rispetto a Trello, dove è possibile sfruttare le feature di uso più comune gratuitamente.

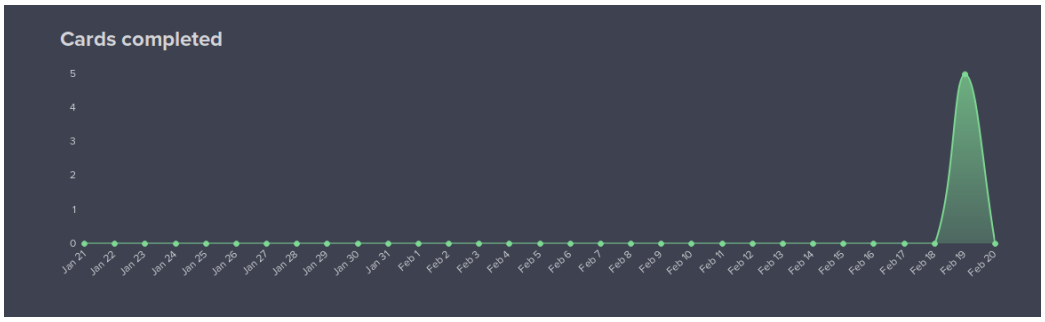

Figura 4.10: Grafico "Cards completed" in UpWave

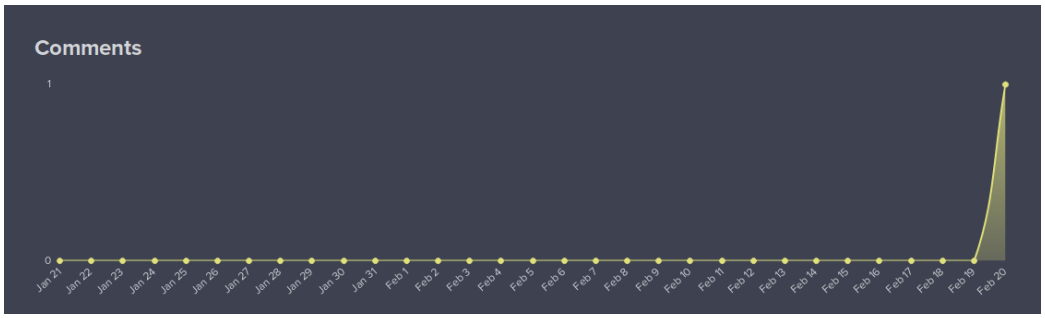

Figura 4.11: Grafico "Comments" in UpWave

4.7 Installazione locale

La natura open-source di Taiga permette agli utenti di avere un maggior controllo sul software, infatti, è possibile installare il progetto su un proprio server per avere un controllo completo sui dati.

I software open-source sono spesso noti per avere una buona documentazione e Taiga ne costituisce un esempio. La sua documentazione è ben strutturata e chiara ed è possibile consultarla alla pagina `https://taigaio.github.io/taiga-doc/dist/`.

Ovviamente, la documentazione comprende una lista dei requisiti necessari per poter installare e far funzionare Taiga. In aggiunta a queste informazioni sono presenti varie modalità di installazione del software.

Una prima distinzione è data dal tipo di ambiente che si vuole creare: produzione o sviluppo. Per entrambi gli ambienti la documentazione elenca i comandi da effettuare e fornisce una copia dei file di configurazione necessari. In questa tesi viene considerata solo l'installazione di un ambiente adatto alla produzione poiché è un caso di interesse più generale.

Oltre all'installazione "manuale" la documentazione fornisce dei metodi alternativi per automatizzare questa fase:

- taiga-scripts: l'installazione viene automatizzata grazie all'esecuzione di script bash; permette di installare Taiga in ambiente di produzione o in quello di sviluppo.
- taiga-vagrant: automatizza il processo di installazione con la creazione di una macchina virtuale con sistema operativo Ubuntu per poi installare su quest'ultima Taiga ed esporlo nella rete locale; questo metodo fa uso a sua volta dei taiga-scripts e installa l'ambiente di produzione.
- taiga-docker: elenca varie immagini per Docker che forniscono un'installazione di Taiga (l'ambiente dipende dall'immagine scelta).
- puppet-taiga: fornisce uno script per il deploy di Taiga su un server sfruttando le pipeline fornite da Puppet.
- taiga-cloudron: il Cloudron Team ha reso disponibile Taiga tra i software che possono essere installati su un server fornito da Cloudron.

Solamente i taiga-scripts e taiga-vagrant sono metodi ufficiali, invece gli altri sono creazioni degli utenti che il team Taiga ha voluto inserire nella documentazione per ampliare l'offerta delle possibili configurazioni per Taiga.

Nelle installazioni automatiche non sono compresi i plugin per le integrazioni con software esterni, ma le istruzioni per aggiungerli sono reperibili sui

relativi repository Github.

L'installazione locale di Taiga non aggiunge feature rispetto a quella resa disponibile sui server Taiga, l'unica differenza consiste nella possibilità di modificare e aggiungere temi per modificarne l'interfaccia.

4.7.1 Un esempio pratico: Exquisite Corpse

Per esporre i concetti illustrati in modo più chiaro è stata fatta un'installazione locale di Taiga e un progetto di esempio che verrà sviluppato usando il metodo Kanban.

Il primo passo consiste nella clonazione del codice sorgente del progetto "taiga-vagrant" e nell'installazione di Virtual Box e Vagrant. Quest'ultimo è un software prodotto dalla HashiCorp che permette di descrivere e configurare delle macchine virtuali tramite un file chiamato "Vagrantfile", che viene scritto in Ruby. Questo metodo è stato preferito agli altri perché mantiene isolata l'installazione di Taiga e delle sue dipendenze; nel caso di errori fatali è possibile distruggere la macchina virtuale e ricrearla senza ripercussioni sul sistema operativo che la ospita.
Una volta terminata la fase di preparazione e installati i software necessari, è possibile cominciare la vera e propria installazione di Taiga: in questa fase Vagrant si occupa di istanziare una macchina virtuale con sistema operativo Ubuntu 16.04 e di eseguire tutti i passaggi necessari per la corretta installazione di Taiga; al termine di questa fase è possibile chiamare la porta 5000 del sistema operativo ospite per essere connessi con la stessa porta del sistema operativo ospitato.

A questo punto è possibile accedere alla propria installazione di Taiga tramite browser. La versione locale di Taiga potrebbe sembrare identica a quella fornita dai server del team Taiga, ma si riscontra da subito una differenza:

la funzione che permette di esplorare i progetti pubblici ha una funzionalità ridotta a causa dell'ambiente isolato, questa funzione permette di cercare solamente tra i progetti pubblici disponibili sul server in cui è stato installato Taiga; la sua utilità a questo punto si circoscrive all'uso di template creati dal team.
Ovviamente, questa limitazione si estende anche agli utenti: possono essere inseriti all'interno del progetto solo gli utenti Taiga presenti nell'installazione locale.

L'idea

Exquisite Corpse è un progetto molto semplice che vuole riprodurre un gioco di epoca surrealista costituito da poche e semplici regole: un gruppo di giocatori compone una poesia o un racconto alternandosi alla scrittura, con il vincolo che ogni partecipante può vedere solo la parola finale del contributo precedente.

Il progetto è costituito da una pagina web in cui un utente può scegliere di contribuire ai racconti già disponibili oppure di cominciarne uno nuovo, in questo caso sceglie anche il numero di parole da cui sarà composto lo stesso. L'utente dovrà essere autenticato per poter partecipare attivamente al gioco, altrimenti sarebbe impossibile controllare che egli non violi le regole scrivendo parti consecutive del racconto.

Si è volutamente scelto un progetto molto semplice per evitare di concentrare troppa attenzione sull'idea e portarla, invece, sul metodo con cui realizzarla.

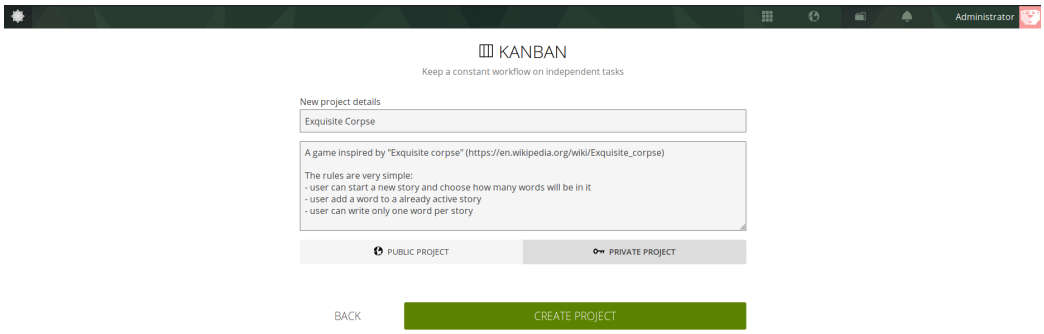

Figura 4.12: Creazione del progetto "Exquisite Corpse" in un'installazione locale di Taiga

Il team

Per la realizzazione del progetto è stato pensato un team di 6 membri organizzati secondo il metodo Kanban.
A un utente è stato assegnato il ruolo di Service Request Manager e a un altro quello di Service Delivery Manager, mentre i rimanenti membri del team sono stati assegnati ai team funzionali Design, UX, Back e Front. In questo caso i team funzionali sono composti da un solo utente e risultano utili solamente per distinguere le capacità di ciascun sviluppatore.

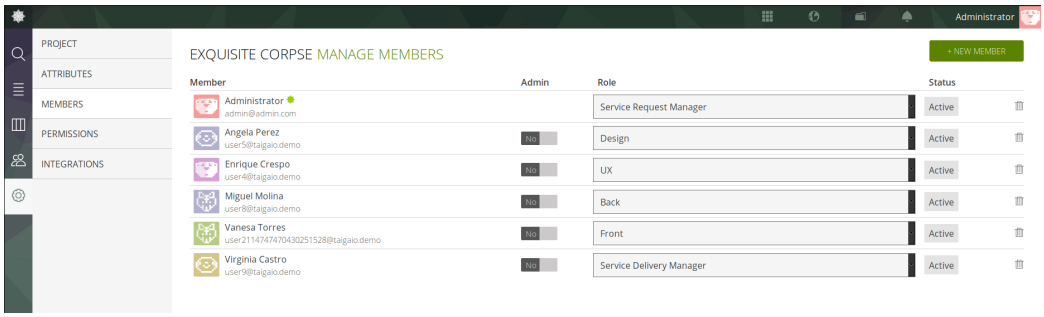

Figura 4.13: I membri del team

Per riprodurre i ruoli tipici di Kanban è stato eliminato il Product Owner, poiché questa nomenclatura è più adatta a Scrum, e sono stati creati

i ruoli di Service Request Manager e Service Delivery Manager: entrambi possono creare, modificare ed eliminare User Stories a piacere e assegnarle ai membri del team, ma non possono modificare le stime fatte dagli sviluppatori.

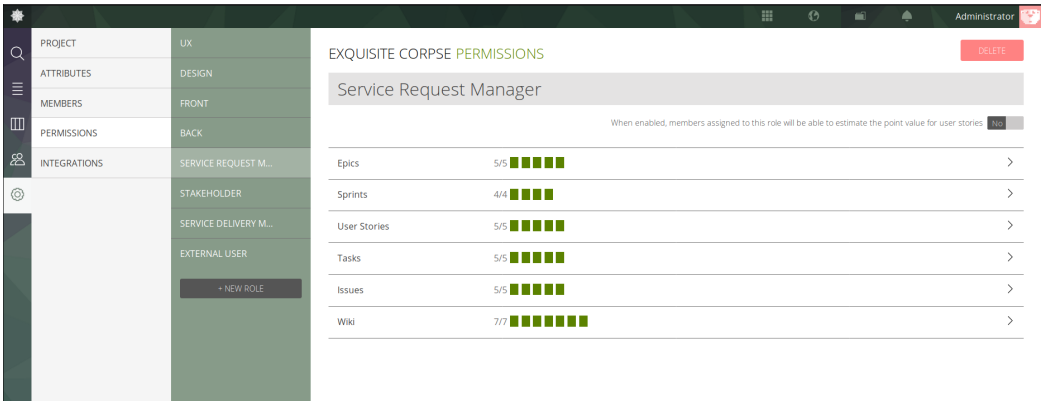

Figura 4.14: Permessi assegnati ai diversi ruoli

Kanban Board

Il team è organizzato secondo il metodo Kanban, quindi il progetto può essere rappresentato tramite una Kanban Board. Quest'ultima è composta dalle colonne:

- New: costituisce il backlog del progetto
- Ready: contiene le prossime User Stories da sviluppare
- In Progress: contiene le User Stories in fase di sviluppo
- Ready for Tests: contiene le User Stories già completate e di cui bisogna testare il corretto funzionamento
- Done: contiene le User Stories completate

Le colonne "Ready", "In Progress" e "Ready for Tests" hanno dei limiti WIP, rispettivamente 4, 6, 4, stabiliti dal team in modo empirico.

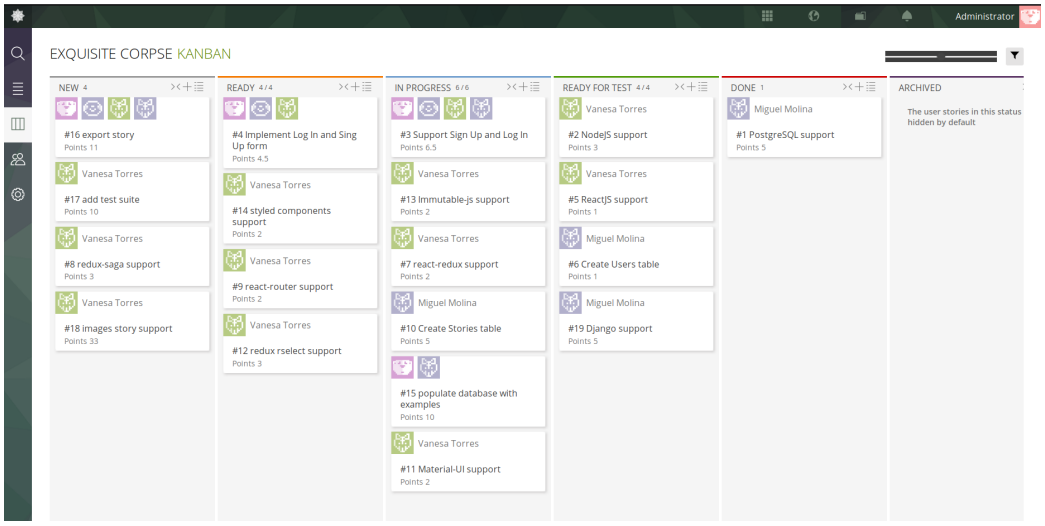

Figura 4.15: Kanban Board del progetto Exquisite Corpse

Conclusioni

In questa tesi sono stati analizzati due metodi, Scrum e Kanban, e un terzo, Scrumban, che permette la transizione dal primo al secondo e diventa a sua volta un metodo a sé stante.

Nell'esposizione dei tre metodi sono stati approfonditi gli strumenti che vengono messi a disposizione del team. In particolare, esiste uno strumento comune a tutti e tre i metodi presentati: la lavagna (fisica o virtuale). Attraverso l'analisi dei diversi utilizzi di quest'ultima è stato possibile enfatizzare le somiglianze e le divergenze tra i metodi.

L'aspetto più interessante derivante da questo studio è la comprensione di come ognuno dei tre metodi gestisca le attività contenute nella lavagna in modo differente. È interessante vedere come una piccola variazione nell'interpretazione della lettura di questo strumento denoti una differenza sostanziale nella filosofia e nell'approccio allo sviluppo software.

Purtroppo non è possibile stilare una classifica dei metodi esposti, ogni team deve valutare in base alla propria filosofia e alla propria gestione dei ruoli quale metodo è più adatto e affine alla propria forma mentis. Questo non significa che un team non potrà mai adottare un metodo che abbia un approccio "nuovo", ma solo che ci vorrà più tempo per comprenderlo e applicarlo. Scrumban è un esempio di come questo processo di transizione possa essere accompagnato per evitare di incorrere in cambiamenti drastici,

che potrebbero confondere i membri del team o portare a una situazione di incomprensione di ciò che sta succedendo.

Inoltre Scrumban costituisce un valido esempio di come diversi metodi possano essere combinati col risultato di crearne di nuovi: usare la struttura fornita da Scrum per applicare gli strumenti di Kanban genera un nuovo processo di sviluppo, che può essere visto o come strumento di transizione o come una nuova impostazione.

Appendice A

Appendice

In questa appendice vengono esposti i comandi per l'installazione e il lancio del servizio Taiga tramite Vagrant. Lo scopo è quello di fornire una descrizione più tecnica dei passaggi descritti nella sezione 4.7.

Per prima cosa è necessario installare i software necessari per il funzionamento di Vagrant. Poiché quest'ultimo è un programma che permette di automatizzare la gestione di una macchina virtuale, è necessario installare un software che si occupi della creazione e della configurazione delle macchine virtuali; qui viene presentata l'installazione di Virtual Box perché è open-source, gratuito e facilmente installabile sui sistemi operativi più diffusi.
Per installare Virtual Box è possibile scaricarlo dal sito ufficiale `https://www.virtualbox.org/` oppure dai repository ufficiali della propria distribuzione linux con il comando seguente[1]:

```
$ sudo apt install virtualbox virtualbox-ext-pack
```

Una volta installato Virtual Box è possibile installare Vagrant, anche questo software è scaricabile dal sito ufficiale `https://www.vagrantup.com/` o

[1] I comandi presentati sono validi esclusivamente per le distribuzioni basate su Debian; per le altre distribuzioni Linux è necessario verificare il nome corrispondente per ciascun pacchetto che si vuole installare all'interno del package manager a disposizione.

dai repository Linux con il comando[2]:

```
$ sudo apt install vagrant
```

Inoltre, per poter clonare il progetto Taiga è necessario aver installato Git, anche questo installabile dal sito ufficiale `https://git-scm.com/` o reperibile nei repository Linux con il comando[3]:

```
$ sudo apt install git
```

Terminata la fase preparatoria, è possibile cominciare l'installazione vera e propria di Taiga. Per prima cosa è necessario clonare il progetto "taiga-vagrant" con il comando seguente[4]:

```
$ git clone https://github.com/taigaio/taiga-vagrant.git
```

e accedere al progetto:

```
$ cd taiga-vagrant
```

Una volta che si è clonato il progetto è necessario istanziare la macchina virtuale tramite Vagrant con il comando seguente[5]:

```
$ vagrant plugin install vagrant-vbguest && vagrant up
```

L'istanziazione della macchina virtuale può richiedere diversi minuti se non si è già in possesso dell'immagine di Vagrant richiesta nel Vagrantfile. Inoltre, questo comando provvede alla configurazione della macchina virtuale per fornire tutti i software necessari per l'esecuzione di Taiga.

Una volta terminata la creazione della macchina virtuale questa viene avviata automaticamente ed è possibile accedere a Taiga all'indirizzo `http://localhost:8000`. Inizialmente si accede come utente *Admin* senza la necessità di autenticarsi, ma se si vuole procede con la creazione di nuovi utenti è necessario fare log-out e creare un nuovo utente; quando si vuole

[2]Vedi nota 1.
[3]Vedi nota 1.
[4]Vedi nota 1.
[5]Vedi nota 1.

rientrare come utente amministratore è necessario inserire le credenziali per username *admin* e password *123123*.

Giunti a questo punto l'installazione di Taiga è correttamente funzionante, anche se sprovvista di alcuni plugin (per esempio non viene installato il modulo per l'integrazione con Slack) che, però, possono essere installati successivamente.
Se si vuole riprodurre un'installazione completa è necessario accedere alla macchina virtuale tramite il comando:

```
$ vagrant ssh
```

e per ogni plugin che si vuole aggiungere seguire le apposite istruzioni, reperibili all'interno della documentazione all'indirizzo `https://taigaio.github.io/taiga-doc/dist/#contrib-plugins`.

Uno dei vantaggi di Vagrant è che per interrompere l'esecuzione di Taiga è sufficiente fermare la macchina virtuale con il comando:

```
$ vagrant halt
```

L'aspetto più interessante, però, è quello di poter distruggere la macchina virtuale in qualsiasi momento con il comando:

```
$ vagrant destroy
```

Una volta rimossa l'installazione precedente è possibile istanziarla nuovamente con il comando:

```
$ vagrant up
```

In questo modo è sempre possibile tornare a uno stato funzionante del software. L'unica controindicazione è quella di perdere tutti i dati inseriti nell'installazione precedente, ma questi possono essere esportati e salvati nella cartella /home/vagrant/data prima di distruggere la macchina virtuale per poterli usare anche in un'installazione successiva o semplicemente per usarli dalla macchina ospite.

Questa soluzione, ovviamente, non si presta tanto per il rilascio sul server quanto per avere una piattaforma uguale per tutti su cui fare test di vario

tipo.

Bibliografia

[1] Corey L., *Scrumban - Essays on Kanban Systems for Lean Software Development*, Modus Cooperandi Press, 2009

[2] Stellman A., Greene J., *Learning Agile - Understanding Scrum, XP, Lean, and Kanban*, O'Reilly Media, 2013

[3] Poppendieck M., Poppendieck T., *Lean Software Development: An Agile Toolkit*, Addison Wesley Professional, 2003

[4] Schwaber K., Sutherland J., *The Scrum Guide - The Definitive Guide to Scrum: The Rules of the Game*, v2017, url: `https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf`

[5] Pahuja S., *What is Scrumban?*, url: `https://www.agilealliance.org/what-is-scrumban/`

[6] Beck K., Beedle M., van Bennekum A., Cockburn A., Cunningham W., Fowler M., Grenning J., Highsmith J., Hunt A., Jeffries R., Kern J., Marick B., Martin R. C., Mellor S., Schwaber K., Sutherland J., Thomas D., *Principles behind the Agile Manifesto*, url: `http://agilemanifesto.org/principles.html`

[7] Klipp P., *Getting Started with Kanban for Software Development*, url: `https://kanbanery.com/ebook/GettingStartedWithKanban.pdf`

[8] Karaivanov D., *The Flow Manager in Kanban*, url: https://www.linkedin.com/pulse/flow-manager-kanban-dimitar-karaivanov

[9] Anderson D., *Emerging Roles In Kanban*, url: https://www.linkedin.com/pulse/emerging-roles-kanban-david-anderson

[10] Anderson D., *When Do We Need SDM & SRM Roles With Kanban?*, url: https://leankanban.com/when-do-we-need-sdm-srm-roles-with-kanban/

[11] Scotland K., *Kanban, Flow and Cadence*, url: https://kanbantool.com/kanban-library/implementing-kanban/kanban-flow-and-cadence

[12] SCRUMstudy, *What is SCRUM Quality?*, url: http://blog.scrumstudy.com/what-is-scrum-quality/

[13] Novkov A., *The Kanban Roles You?ve Probably Never Heard Of*, url: https://dzone.com/articles/the-kanban-roles-youve-probably-never-heard-of

[14] *7 Lean Metrics to Improve Flow*, url: https://leankit.com/learn/kanban/lean-flow-metrics/

[15] Karunanithi K., *Metrics in Agile and Kanban, Software Measurement Techniques*, url: https://www.researchgate.net/publication/305613018_Metrics_in_Agile_and_Kanban_Software_Measurement_Techniques

[16] Hall R., *Scrum Process*, url: https://documentation.cochrane.org/display/WWIRPT/Scrum+process

[17] Radigan D., *Five agile metrics you won't hate*, url: https://www.atlassian.com/agile/project-management/metrics

Ringraziamenti

Ringrazio il prof. Ciancarini per la sua disponibilità e per avermi incentivato ad approfondire gli argomenti della tesi.

Un ringraziamento speciale alla mia famiglia, alla mia fidanzata e alla mia gatta, che mi hanno supportato e sopportato in questo percorso.