

# Sviluppo Agile del Software

Dai principi agili a SCRUM



# Processi di sviluppo agili

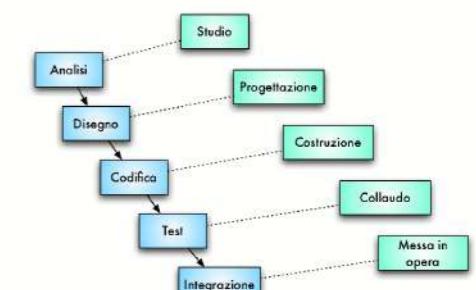
- Process intensive vs sviluppo Agile
- Un processo di progettazione Agile rende il processo abbastanza flessibile da adattarsi facilmente e velocemente ai cambiamenti nelle tecnologie



- SCRUM



Waterfall



# Necessità di sviluppo software rapido

- Il rapido sviluppo e la consegna sono oggi spesso il requisito più importante per i sistemi software
- Le aziende operano in un contesto in rapida evoluzione ed è praticamente impossibile produrre una serie di requisiti software stabili.
- Il software deve evolvere rapidamente per riflettere le mutevoli esigenze aziendali.
- Lo sviluppo pianificato è essenziale per alcuni tipi di sistemi, ma non soddisfa queste esigenze aziendali.
- Metodi di sviluppo agili sono emersi alla fine degli anni '90 con l'obiettivo di ridurre radicalmente i tempi di consegna per i sistemi software di lavoro.

# Necessità di sviluppo software rapido

- software sempre più complessi
- utenti spesso delusi da un prodotto software che dopo mesi non è la soluzione desiderata
- ci siamo focalizzati sull'aspettare e gestire dei tempi e non sull'effettivo obiettivo dell'utente
- abbiamo privilegiato le fasi di progettazione ai contatti con l'utente
- l'utente spesso non è coinvolto nello sviluppo e non può dare un feedback
- si deve identificare un modo per capire cosa è di valore per il nostro utente
- seguire un piano senza confrontarsi o adattarsi porta ad un altissimo grado di fallimento
- AGILE è attualmente il meglio che possiamo avere per ottenere il massimo in funzione di quello che abbiamo a disposizione

# Sviluppo Agile

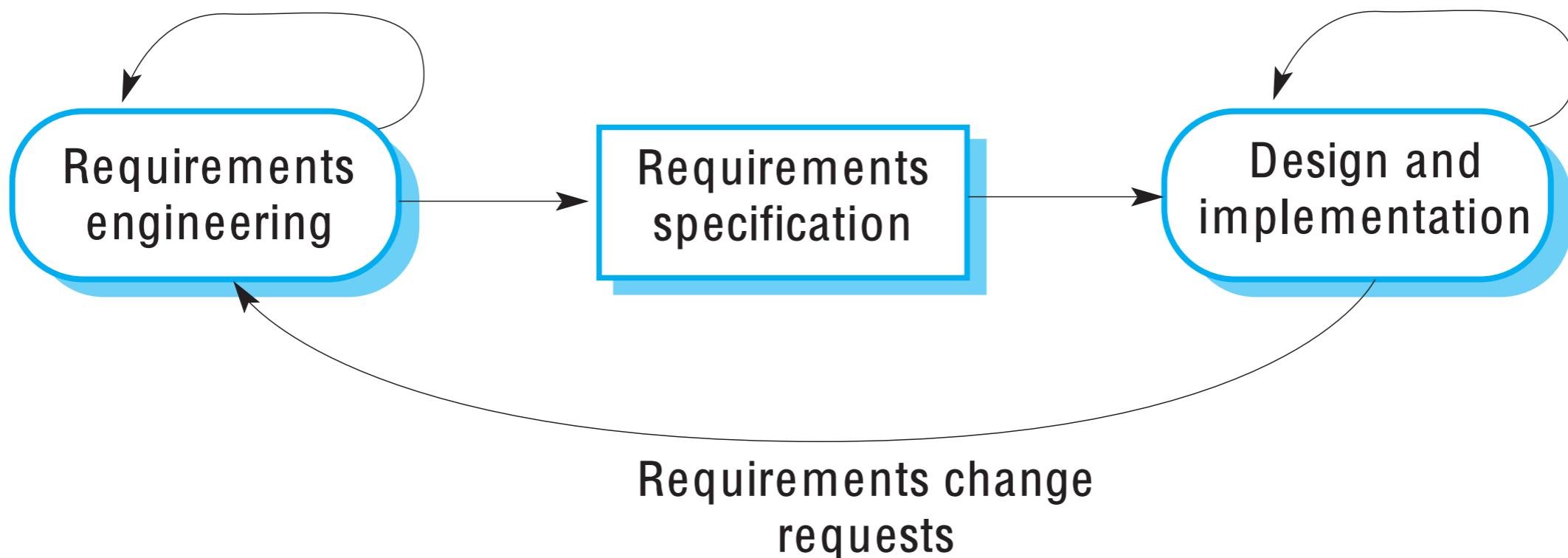
- Le specifiche, la progettazione e l'implementazione del programma sono intrecciate
- Il sistema è sviluppato come una serie di versioni o incrementi con gli stakeholder coinvolti nelle specifiche e nella valutazione della versione.
- Consegna frequente di nuove versioni per la valutazione
- Ampio supporto di strumenti (ad es. strumenti di test automatizzati) utilizzati per supportare lo sviluppo.
- Documentazione minima - focus sul codice di lavoro

# Sviluppo Agile

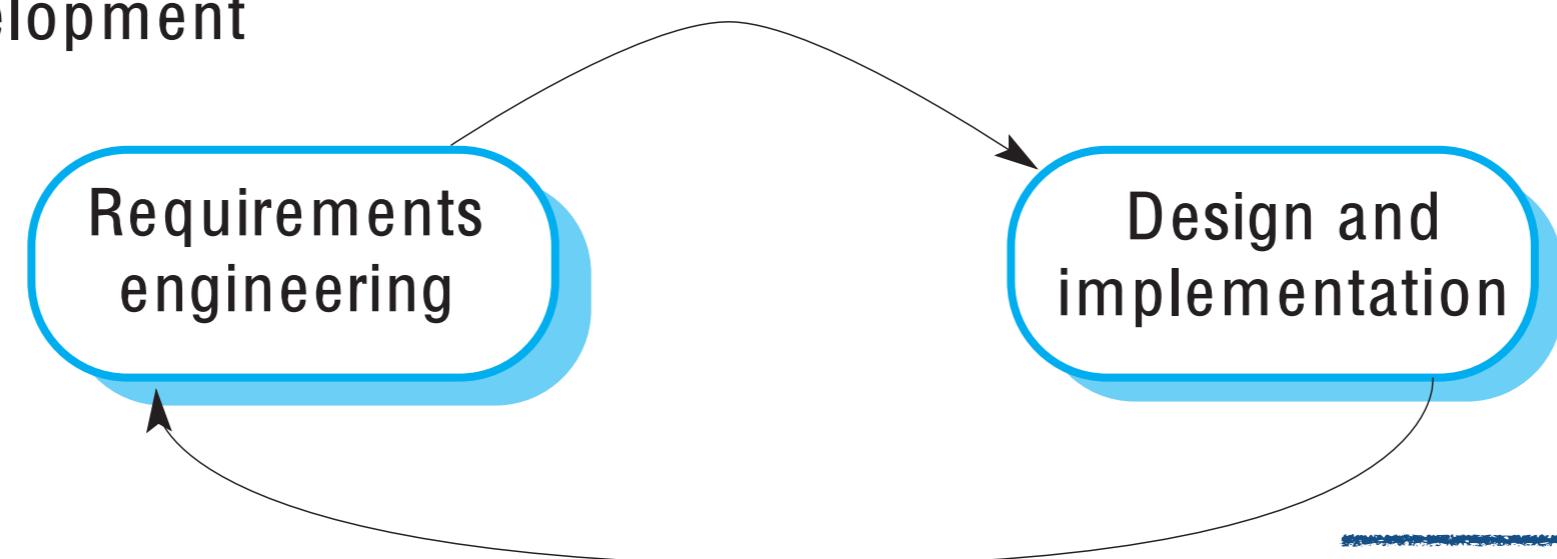
- Tutto viene sintetizzato nel 2001
  - 4 valori fondamentale e 12 principi (Manifesto Agile)
- Agile non è una metodologia di sviluppo, è un approccio basato sui principi dell'agility

# Plan driven vs Agile

## Plan-based development



## Agile development



# Plan driven vs Agile

- Sviluppo guidato dal piano
  - Un approccio all'ingegneria del software basato su un piano si basa su fasi di sviluppo distinte, in cui i risultati devono essere prodotti in ciascuna di queste fasi e pianificati in anticipo.
  - Non necessariamente modello a cascata - sviluppo planimetrico, incrementale possibile
  - L'iterazione si verifica all'interno delle attività.
- Sviluppo agile
  - Le specifiche, la progettazione, l'implementazione e i test sono interdipendenti e i risultati del processo di sviluppo sono decisi attraverso un processo di negoziazione durante il processo di sviluppo del software.
  - Consegnna frequente di nuove versioni per la valutazione
  - Ampio supporto di strumenti (ad es. strumenti di test automatizzati) utilizzati per supportare lo sviluppo.
  - Documentazione minima - focus sul codice di lavoro

# Metodi Agili

- L'insoddisfazione per le spese generali coinvolte nei metodi di progettazione software degli anni 1980 e 1990 ha portato alla creazione di metodi agili. Questi metodi:
  - Si concentrano sul codice piuttosto che sul design
  - Si basano su un approccio iterativo allo sviluppo di software
  - Sono destinati a fornire software in modo rapido ed evolvere rapidamente per soddisfare le mutevoli esigenze.
  - L'obiettivo di metodi agili è quello di ridurre le spese generali nel processo software (ad esempio limitando la documentazione) e di essere in grado di rispondere rapidamente alle mutevoli esigenze senza un'eccessiva rielaborazione.

# Manifesto Agile

- [www.agilemanifesto.org](http://www.agilemanifesto.org)
- fine anni '90

# Manifesto Agile

Stiamo scoprendo modi migliori di creare software sviluppandolo e aiutando gli altri a farlo.  
Attraverso questo lavoro abbiamo acquisito valore:

- Individui e interazioni su processi e strumenti
  - la forza di un'azienda, quello che c'è dentro
- Software funzionante su documentazione completa
  - la documentazione è un valore solo se è richiesto dall'utente
- Collaborazione con i clienti sulla negoziazione di contratti
  - la documentazione è un valore solo se è richiesto dall'utente
  - collaborazione con il cliente più che contratti (un contratto è in genere un qualcosa che serve a tutelare l'utente da comportamenti scorretti) cambio di prospettiva sul concetto di contratto.
  - con il cliente bisogna collaborare, seguire i feedback per ottenere un risultato migliore più che irrigidirsi sul contratto
- Risposta al cambiamento seguendo un piano
  - avere la forza e la capacità di rispondere ai cambiamenti

Cioè, mentre c'è valore nelle voci a destra, diamo più valore alle voci a sinistra.



## Principle

## Description

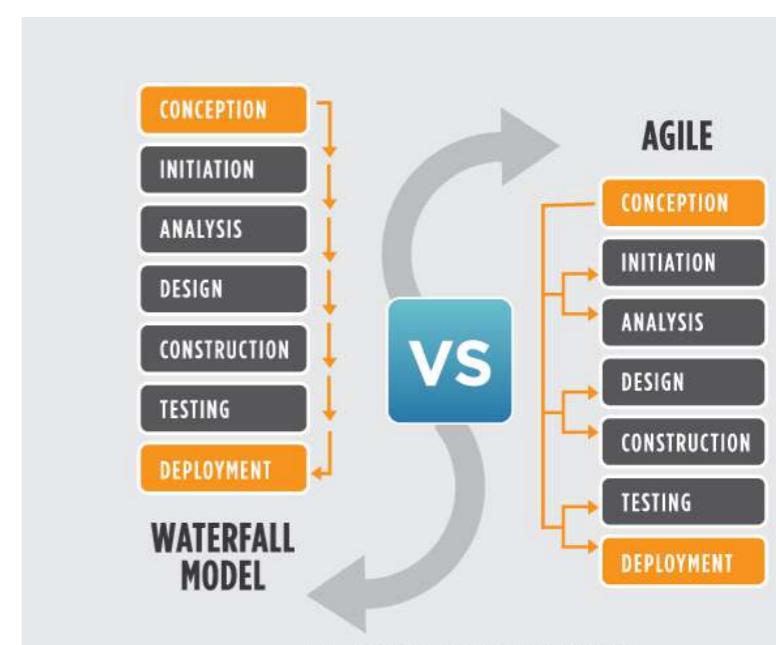
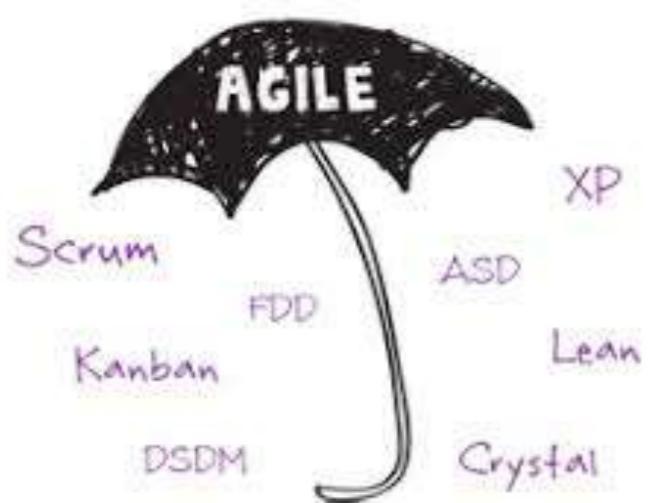
Principle	Description
Coinvolgimento dei clienti	I clienti dovrebbero essere strettamente coinvolti in tutto il processo di sviluppo. Il loro ruolo è quello di fornire e dare priorità ai nuovi requisiti di sistema e di valutare le iterazioni del sistema.
Consegna incrementale	Il software viene sviluppato a intervalli con il cliente, specificando i requisiti da includere in ogni incremento.
Persone non processi	Le competenze del team di sviluppo devono essere riconosciute e sfruttate. I membri del team dovrebbero essere lasciati a sviluppare il proprio modo di lavorare senza procedure prescrittive.
Accettare i cambiamenti	Prevedere che i requisiti di sistema vengano modificati e quindi progettare il sistema in modo che tenga conto di tali modifiche.
Mantenere la semplicità	Concentrarsi sulla semplicità sia nel software in fase di sviluppo che nel processo di sviluppo. Ove possibile, lavorare attivamente per eliminare la complessità dal sistema.

# Applicabilità dei metodi agili

- Sviluppo di prodotti quando un'azienda di software sviluppa un prodotto di piccole o medie dimensioni per la vendita.
- Praticamente tutti i prodotti software e le applicazioni sono ora sviluppati con un approccio agile
- Sviluppo di sistemi personalizzati all'interno di un'organizzazione, dove c'è un chiaro impegno da parte del cliente di essere coinvolto nel processo di sviluppo e dove ci sono poche regole e regolamenti esterni che influenzano il software.



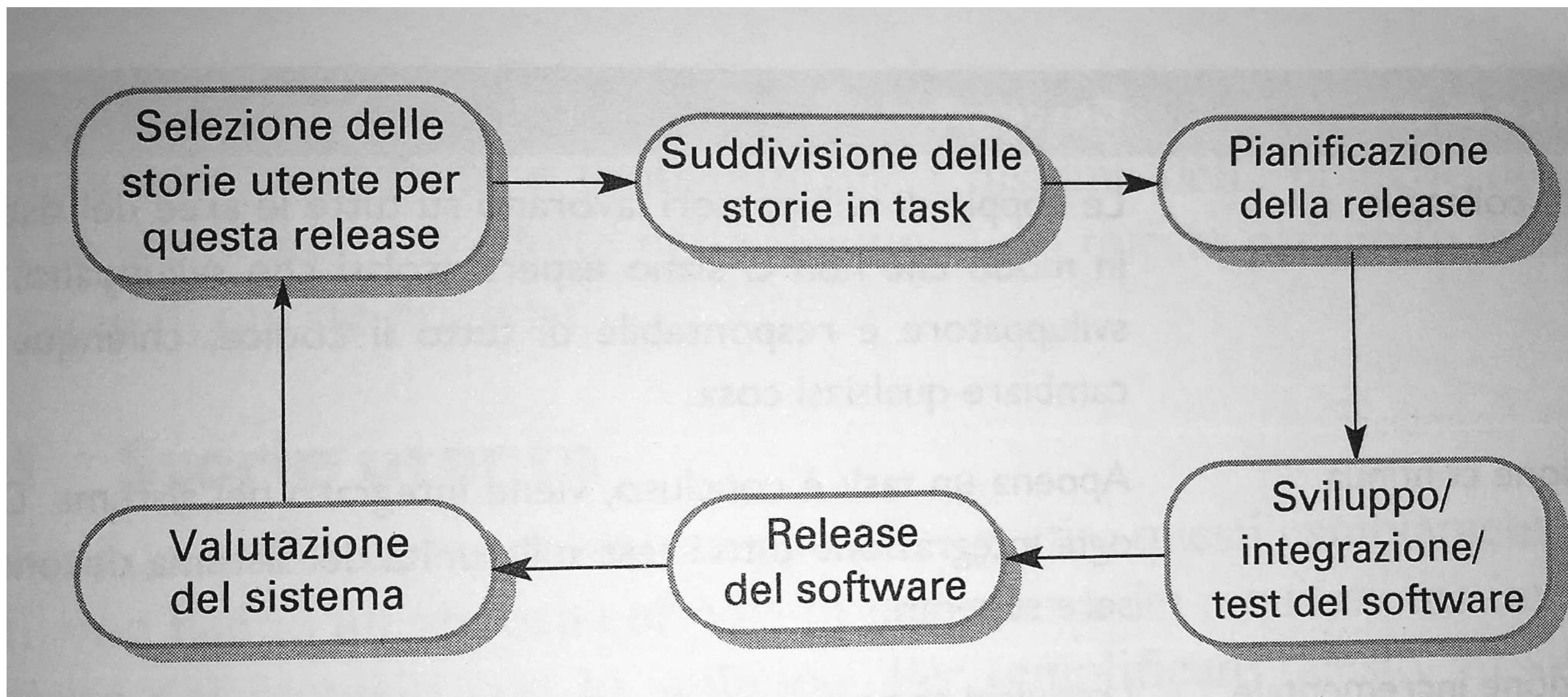
# Tecniche di sviluppo Agile



# Extreme programming

- Un metodo agile molto influente, sviluppato alla fine degli anni '90, che ha introdotto una serie di tecniche di sviluppo agile.
- La programmazione estrema (XP) adotta un approccio "estremo" allo sviluppo iterativo.
  - Le nuove versioni possono essere costruite più volte al giorno;
  - Gli aumenti vengono consegnati ai clienti ogni 2 settimane;
  - Tutti i test devono essere eseguiti per ogni build e la build viene accettata solo se i test vengono eseguiti con successo.

# Il ciclo di rilascio dell'XP



- # Pratiche di programmazione estrema

Principio o pratica	Descrizione
Proprietà collettiva	Le coppie di sviluppatori lavorano su tutte le aree del sistema, in modo che non ci siano esperti isolati che sviluppano; ogni sviluppatore è responsabile di tutto il codice; chiunque può cambiare qualsiasi cosa.
Integrazione continua	Appena un task è concluso, viene integrato nel sistema. Dopo ogni integrazione tutti i test sulle unità del sistema devono essere superati.
Pianificazione incrementale	I requisiti sono registrati su "story card" e le storie da includere in una release sono determinate dal tempo disponibile e dalla loro priorità relativa. Gli sviluppatori suddividono queste storie in "task" di sviluppo (Figure 3.5 e 3.6).
Cliente on-site	Un rappresentante dell'utente finale del sistema (il cliente) dovrebbe essere sempre disponibile per i membri del team XP. In un processo di programmazione estrema, il cliente è un membro del team di sviluppo e ha la responsabilità di consegnare i requisiti del sistema al team per l'implementazione.
Programmazione a coppie	Gli sviluppatori lavorano a coppie, verificando reciprocamente il loro lavoro e fornendo il supporto per fare sempre un buon lavoro.
Refactoring	Tutti gli sviluppatori effettuano in continuazione il refactoring del codice appena si trovano miglioramenti al codice stesso. Questo rende il codice semplice e mantenibile.
Progettazione semplice	Deve essere eseguita la progettazione sufficiente a soddisfare i requisiti correnti, niente di più.
Piccole release	Inizialmente deve essere sviluppato il minimo numero di funzionalità utili. Le release del sistema sono frequenti e aggiungono, in modo incrementale, nuove funzionalità alla release iniziale.
Ritmo sostenibile	Non sono considerati accettabili grandi ritardi, poiché spesso l'effetto finale è la riduzione della qualità del codice e della produttività a medio termine.
Sviluppo con test iniziali	Viene utilizzato un ambiente automatico di test delle unità per provare una nuova parte di funzionalità, prima che questa sia implementata.

# XP e principi “agili”

- Lo sviluppo incrementale è supportato da rilasci di sistema piccoli e frequenti.
- coinvolgimento del cliente significa impegno a tempo pieno con il team.
- Le persone, non il processo, sono supportate dalla programmazione a coppie, la proprietà collettiva del codice e un processo sostenibile che evita lunghe ore di lavoro.
- Modifica supportata da release di sistema regolari.
- Mantenere la semplicità attraverso un costante refactoring del codice.

# Le pratiche di XP

- La programmazione estrema ha un focus tecnico e non è facile da integrare con la pratica di gestione nella maggior parte delle organizzazioni.
- Di conseguenza, mentre lo sviluppo agile utilizza pratiche di XP, il metodo come originariamente definito non è ampiamente usato.
- Pratiche chiave
  - Storie di utenti per specifica
  - Rifattorizzazione (Refactoring)
  - Sviluppo test-first
  - Programmazione di coppie

# User Stories

- In XP, un cliente o un utente fa parte del team XP ed è responsabile di prendere decisioni sui requisiti.
- Le esigenze dell'utente sono espresse come storie o scenari.
- Queste sono scritte su schede e il team di sviluppo li suddivide in compiti di implementazione. Questi compiti sono alla base del calendario e delle stime dei costi.
- Il cliente sceglie le storie da includere nella prossima release in base alle proprie priorità e alle stime del programma.

# User Stories

- Il cliente lavora a stretto contatto con il team
- Il team sviluppa uno scenario in una release successiva del software
- Il team può suddividere le storie in task
  - si discute con il cliente per perfezionare i requisiti
- Il cliente elenca l'ordine di priorità delle storie
- Si inizia da quelle che possono fornire supporto utile all'azienda
- Se i requisiti variano, le storie non implementate possono essere cambiate o buttate

# User Stories

- Possono essere utili per coinvolgere gli utenti nel suggerire requisiti
- Le persone in genere trovano semplice relazionarsi con queste storie
- Problema principale: incompletezza, è difficile stabilire se tutte le storie sviluppate coprono tutti i requisiti di un sistema
- E' difficile stabilire se la singola storia definisce la rappresentazione vera di una attività

# User story: prescrizione dei farmaci

## Prescrizione dei farmaci

Kate è un dottore che desidera prescrivere un farmaco a un paziente che frequenta una clinica. La cartella clinica del paziente è già visualizzata sul suo computer, quindi fa clic sul campo del farmaco e può selezionare “farmaco corrente”, “nuovo farmaco” o “formulario”.

Se sceglie “farmaco corrente”, il sistema le chiede di controllare la dose. Se Kate vuole cambiare la dose, digita la nuova dose e conferma la prescrizione.

Se sceglie “nuovo farmaco”, il sistema suppone che Kate conosca quale farmaco prescrivere. Kate digita le prime lettere del nome del farmaco. Il sistema visualizza una lista di possibili farmaci che iniziano con quelle lettere. Kate seleziona il farmaco richiesto e il sistema risponde chiedendole di verificare se il farmaco selezionato è corretto. Lei digita la dose e conferma la prescrizione.

Se sceglie “formulario”, il sistema visualizza una finestra di ricerca per il formulario approvato. Kate può cercare il farmaco richiesto e poi lo seleziona. Il sistema le chiede di verificare se il farmaco scelto è corretto. Lei digita la dose e conferma la prescrizione.

Il sistema controlla sempre che la dose sia entro i limiti consentiti; in caso contrario, chiede a Kate di cambiare la dose.

Dopo che Kate ha confermato la prescrizione, dovrà controllarla e poi fare clic su “OK” o su “Cambia”. Se fa clic su “OK”, la prescrizione viene memorizzata nel database di controllo. Se fa clic su “Cambia”, il sistema riavvia il processo di prescrizione dei farmaci.

# Suddivisione in task

**Task 1: cambia la dose del farmaco prescritto**

**Task 2: scelta dal formulario**

**Task 3: controllo della dose**

Il controllo della dose è una precauzione per verificare che il dottore non abbia prescritto una dose pericolosamente piccola o grande.

Utilizzando il codice del formulario come nome del farmaco, ricerca nel formulario le dosi minima e massima consigliate.

Confronta la dose prescritta con la minima e la massima. Se la dose è fuori dal range ammesso, visualizza un messaggio di errore per segnalare che la dose prescritta è troppo piccola o troppo grande. Se è all'interno del range, abilita il pulsante “Conferma”.

**Figura 3.6** Esempi di carte di task per prescrivere i farmaci.

# Refactoring

- Precetto fondamentale nell'ingegneria del software è progettare per il cambiamento.
  - vale la pena dedicare tempo e sforzi ad anticipare i cambiamenti, in quanto ciò riduce i costi in una fase successiva del ciclo di vita.
- XP, tuttavia, ritiene che ciò non sia utile in quanto non è possibile prevedere in modo affidabile i cambiamenti.
- Piuttosto, propone un costante miglioramento del codice (refactoring) per rendere più facili le modifiche quando devono essere implementate.

# Refactoring

- Il team di programmazione ricerca possibili miglioramenti software e fa questi miglioramenti anche dove non vi è alcuna necessità immediata.
- Questo migliora la comprensibilità del software e riduce la necessità di documentazione.
- Le modifiche sono più facili da effettuare perché il codice è ben strutturato e chiaro.
- Tuttavia, alcuni cambiamenti richiedono il refactoring dell'architettura e questo è molto più costoso

# Esempi di refactoring

- Riorganizzazione di una gerarchia di classi per rimuovere il codice duplicato.
- Riordinare e rinominare attributi e metodi per renderli più comprensibili.
- Sostituzione del codice in linea con chiamate ai metodi che sono stati inclusi in una libreria di programmi
- quindi miglioramento del codice “su se stesso”

# Sviluppo con test iniziali

- Una delle già grandi differenze tra lo sviluppo incrementale e lo sviluppo guidato da piani è il modo in cui il sistema viene testato
- Nello sviluppo incrementale non c'è una specifica del sistema che può essere usata per creare piano di test
  - al contrario di uno sviluppo plan-driven
- Tuttavia XP ha sviluppato un nuovo approccio al test dei programmi per superare alcune difficoltà tipiche dei test senza specifica.

# Sviluppo con test iniziali

- I test sono fondamentali per XP e XP ha sviluppato un approccio in cui il programma viene testato dopo ogni modifica apportata.
- Funzionalità di test XP:
  - Sviluppo test-first (con test iniziali).
  - Sviluppo di test incrementali a partire da scenari.
  - Coinvoltimento degli utenti nello sviluppo e nella convalida dei test.
  - Test automatizzati vengono utilizzati per eseguire tutti i test dei componenti ogni volta che viene costruita una nuova release.

# Sviluppo con test iniziali

- I test prima del codice chiariscono i requisiti da implementare.
- I test vengono scritti come programmi piuttosto che come dati, in modo da poter essere eseguiti automaticamente. Il test include la verifica della corretta esecuzione.
- Di solito si basa su un framework di test come JUnit.
- Tutti i test precedenti e nuovi vengono eseguiti automaticamente quando viene aggiunta una nuova funzionalità, verificando così che la nuova funzionalità non abbia introdotto errori.

# Descrizione di un test-case

## Task 4: controllo della dose

### Input:

1. Un numero in mg rappresenta una singola dose di farmaco.
2. Un numero rappresenta il numero di singole dosi giornaliere.

### Test:

1. Controlla l'input nel quale la singola dose è corretta, ma la frequenza è troppo alta.
2. Controlla l'input nel quale la singola dose \* la frequenza è troppo grande o troppo piccola.
3. Controlla l'input nel quale la singola dose \* la frequenza è nel range consentito.

### Output:

OK o messaggio di errore per segnalare che la dose è fuori dal range consentito.

**Figura 3.7** Descrizione del test case per controllare la dose del farmaco.

... come che

# Problemi con i metodi agili

- In alcune aree di sviluppo software lo sviluppo Agile ha avuto un successo enorme
- E' indubbiamente il miglior approccio da usare per certi tipi di sistemi
  - prodotti ed applicazioni software
- I metodi agili possono **non** essere appropriati per per sviluppo di sistemi integrati o sistemi grandi e complessi

# Problemi con i metodi agili

- L'informalità di uno sviluppo agile è incompatibile con l'approccio legale alla definizione dei contratti comunemente usato nelle grandi aziende.
- I metodi agili sono più appropriati per lo sviluppo di nuovi software piuttosto che per la loro manutenzione. Tuttavia, la maggior parte dei costi del software nelle grandi aziende deriva dalla manutenzione dei sistemi software esistenti.
- Metodi agili sono progettati per piccoli team co-locati, ma molto sviluppo software coinvolge ora team distribuiti in tutto il mondo.

# Problemi con i metodi agili

- La maggior parte dei contratti software per sistemi personalizzati si basano su una specifica che stabilisce cosa deve essere implementato dal progettista per il cliente del sistema.
- Tuttavia, questo impedisce di interlacciare la specifica e lo sviluppo come è la norma nello sviluppo agile.
- È richiesto un contratto che paghi il tempo dello sviluppatore piuttosto che la funzionalità.
  - Tuttavia, questo è considerato un rischio elevato per i numerosi servizi giuridici, perché ciò che deve essere consegnato non può essere garantito.

# Per quanto riguarda la documentazione

- Mancanza di documentazione del prodotto
  - La documentazione formale serve a descrivere il prodotto
  - serve alle persone che devono modificarlo
  - raramente viene aggiornata e corrisponde al codice
- mantenere il coinvolgimento del cliente
- continuità del team di sviluppo
- Questi sono alcune delle critiche mosse dai sostenitori dell'Agile ai metodi tradizionali

## Prof. Sommerville:

- La mia esperienza di manutenzione dei sistemi indica il documento più importante è il documento dei requisiti del sistema, che spiega all'ingegnere del software che cosa dovrebbe fare il sistema.
- Molti metodi agili raccolgono i requisiti in modo informale e incrementale, non creano un documento dei requisiti coerenti con lo stato reale del sistema.
- L'uso dei metodi agili può quindi rendere più difficile e costosa la successiva manutenzione del sistema. Questo problema è particolarmente grave se non può essere garantita la continuità del team di sviluppo.
- Inoltre non è sempre possibile che il cliente sia presente durante la fase di manutenzione del prodotto

# Relativamente ai tipi di sistema

- I metodi agili furono sviluppati e perfezionati in progetti per sviluppare sistemi aziendali di **piccole e medie dimensioni** di prodotti software, dove lo sviluppatore del software controlla la specifica del sistema.
- Altri tipi di sistemi hanno attributi, quali la dimensione, la complessità, la risposta in tempo reale, le normative esterne, che un puro approccio agile non sarebbe in grado di gestire.
- Per esempio un processo di ingegneria dei sistemi richiede un certo grado di pianificazione, progettazione e documentazione

# Relativamente ai tipi di sistema

- Quanto è grande il sistema da sviluppare?
  - I metodi agili sono più efficaci quando il sistema può essere sviluppato con un team relativamente piccolo.
  - Questo potrebbe non essere possibile per grandi sistemi che richiedono tempi di sviluppo più grandi
- Che tipo di sistema si sta sviluppando?
  - I sistemi che richiedono molta analisi prima dell'implementazione (per esempio i sistemi in tempo reale con requisiti complessi) di solito hanno bisogno di un progetto abbastanza dettagliato per svolgere l'analisi

# Relativamente ai tipi di sistema

- Qual è la vita prevista del sistema?
  - I sistemi di lunga durata possono richiedere più documentazione per comunicare le intenzioni originali degli sviluppatori del sistema al team di supporto.
  - Tuttavia, i sostenitori dell'agile sostengono che è la documentazione spesso non viene aggiornata quindi non è utile nella fase di manutenzione.
- Il sistema è soggetto ha norme esterne?
  - Se un sistema deve essere approvato da un'autorità esterna di controllo allora sarà necessario produrre una documentazione dettagliata sulla sicurezza del sistema.

# Altri elementi

- Quanto sono bravi i progettisti e i programmati del team di sviluppo?
  - Alcuni sostengono che per i metodi agili servono per progettisti e sviluppatori molto bravi.
- Come è organizzato il team di sviluppo?
  - È distribuito e parte dello sviluppo è affidato a società esterne? In questo caso potrebbe essere utile una buona documentazione.
- Quali tecnologie sono disponibili a supporto dello sviluppo del sistema?
  - I metodi agili spesso si basano su buoni strumenti per tenere traccia dell'evoluzione di un progetto.



# SCRUM



# SCRUM



- Si basa su tre semplici punti: Sprint, Backlog e Scrum Meeting.
  - Molto simile ad extreme Programming prevede di dividere il progetto in blocchi rapidi di lavoro (Sprint)
    - alla fine si consegna una versione al cliente,
    - indica come definire i dettagli del lavoro da fare nell'immediato futuro (Backlog) per averne una definizione estesa,
    - organizza riunioni giornaliere del team di sviluppo (Scrum Meeting) per verificare cosa si è fatto e cosa si farà.

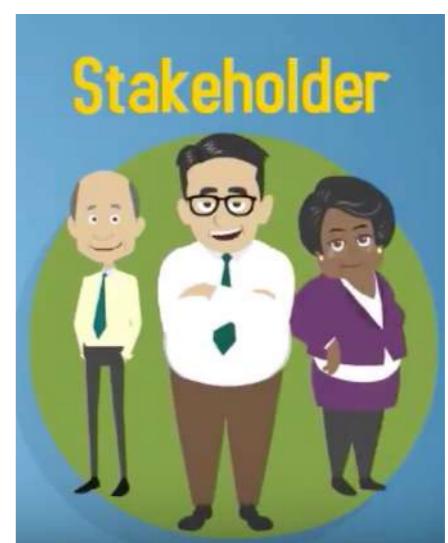


# SCRUM

- E' un framework agile di sviluppo del software concepito per gestire progetti e prodotti software
- Per spiegare SCRUM dobbiamo partire da un'idea di prodotto
  - dobbiamo ricevere le specifiche dagli stakeholder
    - le user stories

# Stakeholder

- tutte le persone interessate dallo sviluppo e uso del prodotto stesso
- clienti, sviluppatori, venditori
  - le persone per le quali il prodotto produce i benefici concordati e ne permettono la produzione



# User stories

- Sono un breve descrizione di una caratteristica detta e scritta dal punto dell'utente che la richiede
- Una user story segue il template a destra
  - as a tipo di utente
  - I want qualche obiettivo
  - so that qualche motivo, qualche ragione

As a  
<type of user>,  
I want  
<some goal>  
so that  
<some reason>.

# User stories

- Consideriamo di essere un creatore di video youtube
- Una user story potrebbe essere

<<come creatore voglio caricare un mio video in modo che tutti possano vederlo>>



# Product Backlog

- Tutte le storie definite vengono raggruppate nel product Backlog
  - fa parte dell'information radiator



tutti i “radiatori” di informazioni devono essere messi in posti visibili a tutti e devono fornire immediatamente tutte le informazioni a tutti quelli che ne hanno bisogno

# I ruoli in SCRUM

- Per lo sviluppo di un prodotto (dettagliato dalle user stories del backlog) c'è bisogno di persone
- in SCRUM le persone necessarie per lo sviluppo di un prodotto sono
  - product owner
  - scrum master
  - team

# Product Owner

- Rappresenta gli stakeholder
  - è la voce del cliente
  - assicura che il prodotto sviluppato raggiunga il valore desiderato
  - scrive le user stories
  - assegna loro una priorità
  - le inserisce nel product backlog

# Scrum Master

- È responsabile della rimozione degli ostacoli
  - che limitano la capacità del team a raggiungere gli obiettivi dello sprint e gli incrementi previsti
  - è un ruolo manageriale ma non è il leader
  - è colui che facilita una corretta esecuzione del processo
  - detiene l'autorità relativa alla applicazione delle norme
  - spesso presiede le riunioni importanti
  - pone sfide alla squadra per migliorarla

# Scrum master

- ruolo fondamentale è di proteggere il team di sviluppo
- funge da cuscinetto verso le distrazioni e protegge il team di sviluppo dalle influenze esterne
- il suo ruolo viene anche denominato "servant leader" per rafforzare queste due prospettive

# Team di sviluppo

- È responsabile della consegna del prodotto con incrementi di caratteristiche tali che il prodotto sia rilasciare alla fine di ogni sprint
- un team di sviluppo è composto da tre a nove persone con competenze tali e trasversali per portare avanti il lavoro effettivo
  - progettazione e sviluppo, test, documentazione tecnica etc.
  - si auto-organizza

# Eventi SCRUM

- Questi sono gli eventi che caratterizzano il framework SCRUM



# Sprint

- Lo Sprint è l'unità di base dello sviluppo SCRUM
  - è di durata fissa - time box- da una a quattro settimane
  - durante lo sprint il team sviluppa le user stories che si è impegnato a rilasciare alla fine dello sprint
    - è definito l'incremento

# Eventi SCRUM

- Gli eventi fondamentali sono 6:
  - Backlog refinement
  - Sprint planning
  - Sprint
  - Daily scrum
  - Sprint review
  - Sprint retrospective

# Backlog Refinement

- Durante il Backlog Refinement il Product Owner incontra il Team per discutere delle storie presenti nel Product Backlog
- Condivide con il Team le priorità da assegnare alle user stories
- Il team aiuta il PO a stimare i costi ed i rischi legati alle user stories
- La stima dei costi è compito del team
  - è il team che ha le competenze per stabilire stima dei tempi e dei costi

# Backlog Refinement

- Sulla base delle stime il PO raffina il Product Backlog
- Seleziona le stories che formeranno il Release Backlog
- Durante la stessa riunione si definisce il concetto di DONE
  - La regola che bisogna rispettare affinché la user story si possa considerare finita
  - La durata di questo evento dipende dalla durata dello sprint
    - all'inizio il tempo impiegato è circa del 15% del totale
    - man mano che si va avanti si attesta al 5-10%

# Backlog Refinement

- La stima delle user stories è sul costo e non sul valore
- Le user storie possono essere stimate in
  - Story point
  - ideal days
  - Ore lavorate
- Consideriamo le ora lavorate

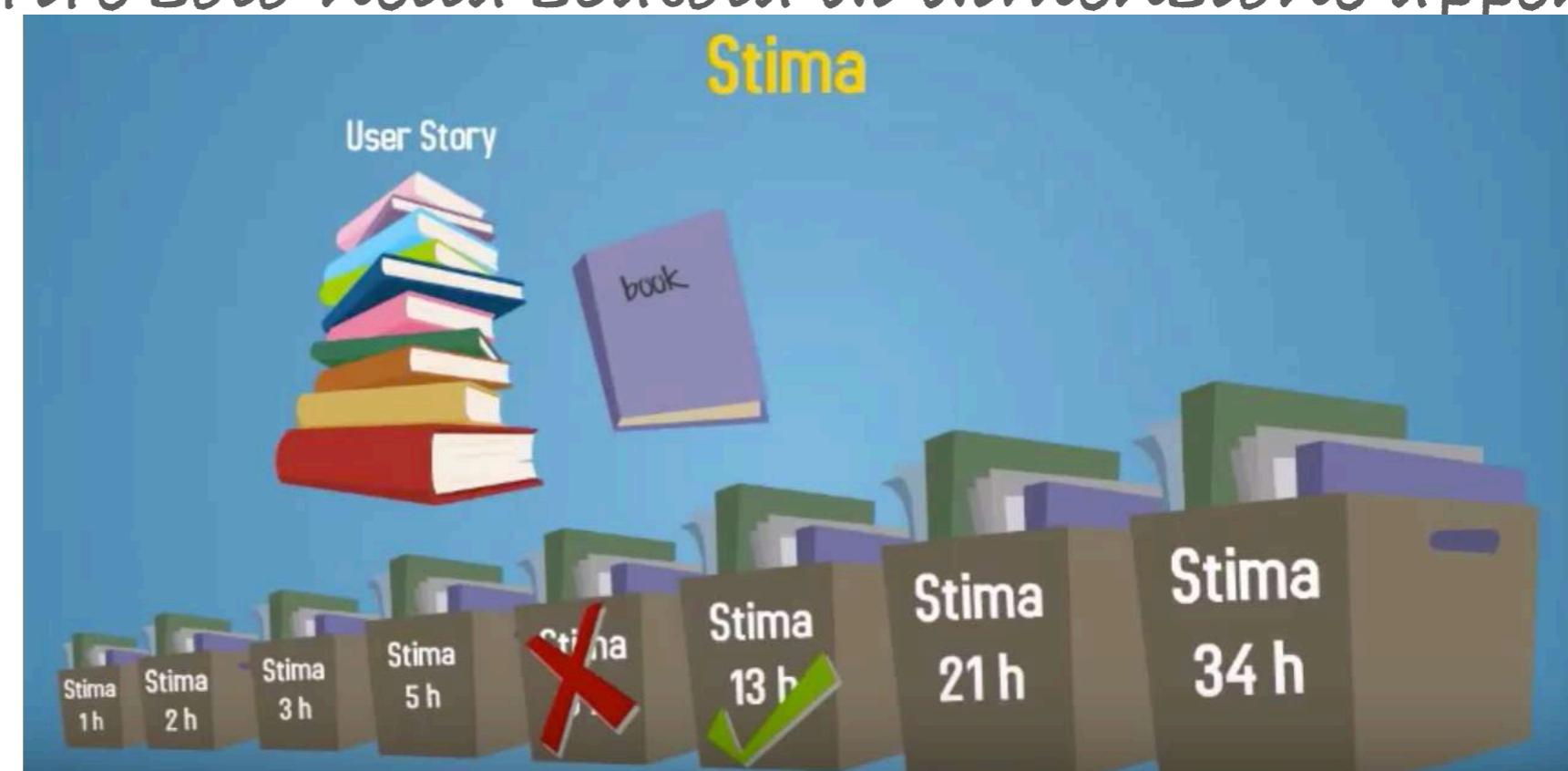
# Backlog Refinement

- I valori da assegnare alle stime dovrebbero seguire la sequenza di Fibonacci



# Backlog Refinement

- Consideriamo le stime come degli scatoloni e le user stories come degli oggetti da inserire dentro
- Se per esempio penso che un'attività ha dimensione 10h la posso inserire solo nella scatola di dimensione appena più grande



# Sprint Planning

- All'inizio di ogni sprint viene tenuto uno sprint planning meeting
  - si pianifica il lavoro da svolgere durante lo sprint
  - partecipano il product owner, il team e lo scrum master
- è un evento time boxed e dura circa due ore per ogni settimana di sprint
- a questo evento il Product Owner si presenta con un backlog con tutta una serie di storie pronte
  - con una priorità assegnata dal product owner

# Sprint Planning

- una norma è che il Product Owner abbia abbastanza storie per almeno due sprint prima di presentarsi ad uno Sprint Planning meeting
- Il team sceglie le storie che si impegna a consegnare nello sprint successivo
  - in base alla priorità
  - ed al numero di storie che il team è in grado di terminare all'interno dello Sprint
- Tutte le storie finiscono sullo sprint Backlog dove le storie possono essere ulteriormente divise in task

# Sprint Planning



# Sprint Planning

- Altro artifact è lo Sprint Goal
  - cosa verrà fatto durante lo sprint
  - scritto in collaborazione tra il Product Owner e il team
- esempio: implementa un basile carrello della spesa con le funzionalità di rimuovere e modificare gli elementi del carrello
- scopo è fornire una breve descrizione, cosa verrà prodotto durante lo sprint

# Sprint Planning

- Durante lo sprint il team lavora per consegnare le user stories che si è impegnato a sviluppare durante lo sprint planning meeting
- gli incrementi, cioè le user stories completate dovranno rispettare la definizione di done
- durante gli sprint il PO controlla le user stories completate e le definisce done se rispettano la definizione
- al termine dell sprint il valore in story point va ad aggiornare la burndown chart e contribuisce a determinare la velocità del team

# Sprint Planning

- La definizione di DONE deve essere condivisa
- prima che il risultato venga rilasciato dev ripetere al definitore condivisa di done
- E' una chiara lista di specifiche che l'incremento deve soddisfare per ritenersi rilasciabile

La definition di done non è statica e si può evolvere nel tempo



# Il Daily Scrum

- Al Daily scrum partecipa il team e lo Scrum Master
  - il PO può partecipare ma non è interessato
- è un evento che si svolge in 15 minuti
- si svolge in piedi
- ogni elemento del team risponde a 3 domande



# Il Daily Scrum

- Il team raggiunge una maggiore comprensione su quanto è stato fatto e quanto rimane da fare
- I membri del team si aggiornano
- non è una riunione di problem solving
  - se ci sono problemi che emergono verranno discussi al termine della riunione con le persone interessate

# La release Burndown chart

- asse orizzontale contiene il numero degli sprint
- asse verticale gli story points o l'unità di misura adottata per stimare le stories
- è un grafico con andamento decrescente
- ad ogni sprint lo scrum master sottrae dal computo degli story point mancanti le storie sviluppate durante lo sprint corrente
- questo grafico serve per calcolare la velocità del team
  - per velocità si intende il numero di story point che il team è in grado di completare durante un'interazione

# Sprint review

- Durante questo incontro il team scrum mostra a tutti gli interessati il risultato prodotto entro uno sprint
- una story/release funzionante
- anche tramite demo, senza presentazione
- solo prodotto funzionante con un time box di un'ora per ogni settimana di sprint
- Il progetto è valutato sulla base degli sprint goal

# Sprint retrospective

- Focalizzata sul processo
- Identifica le cose che il team fa bene ed deve continuare a fare
- Cosa bisogna fare per migliorare
- Cosa non si deve più fare
- Non partecipa il PO ma solo lo Scrum Master
- Questo processo rientra nella capacità di adattamento del team
- Anche qui time-box di un'ora

e dopo....

...e si ricomincia



# Riferimenti

- <http://agilemanifesto.org>
- <http://agilemanifesto.org/principles.html>
- <https://www.scrumalliance.org>
- [https://www.scrumguides.org/docs/scrumguide/v1/  
Scrum-Guide-ITA.pdf](https://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-ITA.pdf)
- <https://www.agilealliance.org/glossary/xp/>