Ingegneria del Software
Metodologie Agile ed eXtreme Programming

Obiettivi.
Presentare le caratteristiche delle metodologie iterative.
Metodo predittivo vs metodo adattivo.
Agile ed eXtreme Programming.

Fulvio Sbroiavacca

Sviluppo del software

- Lo sviluppo software è un **processo complesso**
  - implica l'interazione tra committenza e produttore (membri del team, società esterne o freelance)
  - devono essere definiti i requisiti del prodotto
  - deve essere realizzato il prodotto richiesto con le tecnologie, il tempo e le risorse economiche a loro disposizione
- Si tratta di un sistema complesso e dinamico composto da **molteplici agenti** che agiscono in parallelo e si influenzano reciprocamente
  - richiede feedback, cooperazione e scambio continuo di informazioni: le attività di ogni agente coinvolto nel processo dipendono ed influenzano le attività degli altri agenti
  - un piccolo errore da parte di un agente può generare problemi a tutto il sistema e compromettere il risultato del processo
  - l’elevato numero di elementi coinvolti nel processo comporta difficoltà nello stimare preventivamente tempi e costi necessari alla realizzazione del sistema

Principali categorie di metodologie per lo sviluppo del software (1)

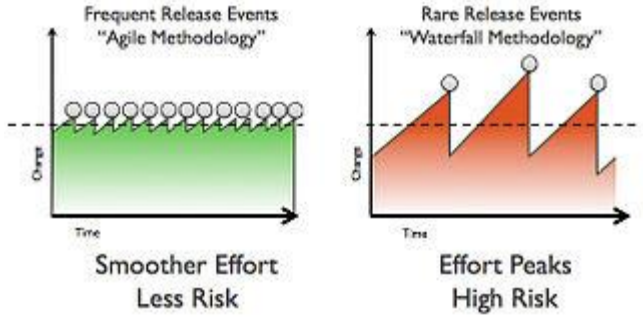
- **Modello a cascata (Waterfall)**
  - Progetto suddiviso in fasi sequenziali (analisi requisiti, disegno applicazione, codifica applicativo, test funzionale, rilascio in produzione)
- **Modello a cascata ibrido (Hybrid Waterfall)**
  - Progetto suddiviso in fasi che possono essere sovrapposte
- **Modello incrementale**
  - Consegna ciclica di insiemi ridotti di funzionalità
- **Modello iterativo**
  - Progetto costituito da tanti brevi cicli sottoposti all'approvazione del cliente
- **Agile**
  - Insieme di metodologie incrementali ed iterative

Principali categorie di metodologie per lo sviluppo del software (2)

- Nei modelli a cascata
  - Utente coinvolto durante la raccolta dei requisiti ed il collaudo per l'accettazione finale
  - Utente che racconta esigenze all'analista ed approva l'analisi ed il disegno
  - Produttore realizza applicazione e la sottopone al collaudo finale
- Nei modelli iterativi
  - Si procede per piccoli cicli: **sprint**
  - Requisiti raccolti, discussi, valutati all'inizio di ogni ciclo coinvolgendo sempre l'utente
  - **Utente parte integrante del team di progetto**
  - L'analisi dei requisiti è distribuita sull'intero arco del progetto con incrementi dipendenti dai risultati conseguiti
  - Team di progetto ed utente collaborano continuamente incontrandosi per condividere le scelte

Metodologie Agile vs metodologie classiche (1)

- Le metodologie classiche tendono ad analizzare un progetto nella sua totalità, per prevedere funzionalità future, con un grande sforzo nella fase di analisi
- Le metodologie “agili” tendono a considerare **piccole parti** del progetto ed a completarle **velocemente**, ottenere subito risultati tangibili, per completare il progetto attraverso **iterazioni successive**
- Nel modello a cascata ogni fase richiede che la precedente sia completata, le continue richieste di variazioni dell’utente o del cliente, richiedono spesso che si debba tornare a fasi precedenti per meglio definire alcuni requisiti
- Si utilizza in questi casi un **processo iterativo** che tende a ripetere più volte le fasi



Effetto della metodologia agile nell'incrementare la frequenza degli eventi di rilascio, spesso misurati in giorni o settimane, in contrasto a grossi, rari rilasci, misurati in quadrimestri o anni, con le tradizionali metodologie di sviluppo

Metodologie Agile vs metodologie classiche (2)

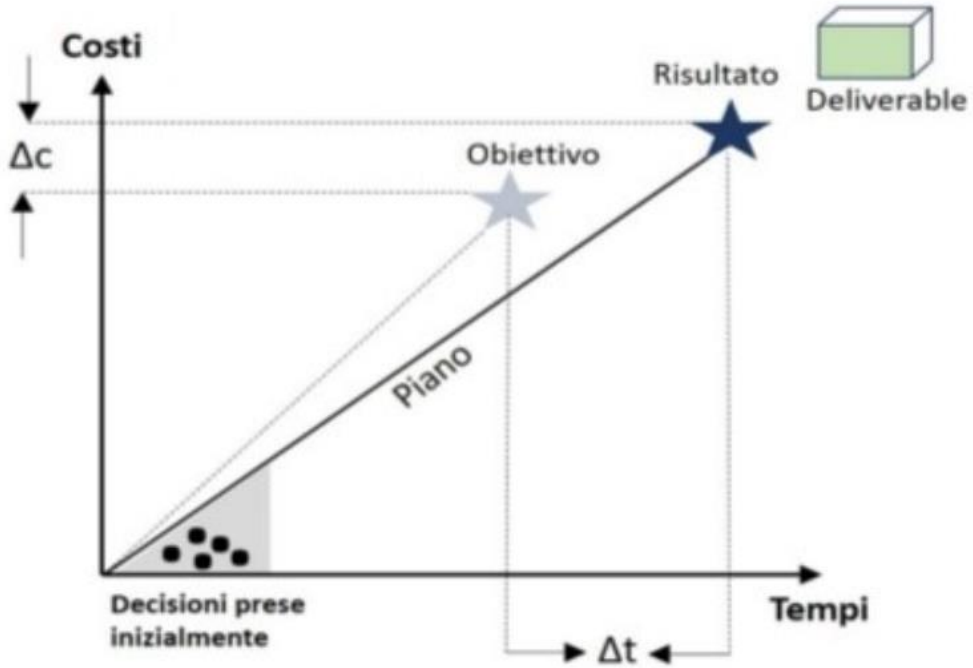
- Nelle metodologie classiche il processo di analisi si propone di prevedere le funzionalità necessarie (**predittivo**)
- Nelle metodologie Agile il processo di analisi mira all'adattabilità in futuro del progetto in base alle necessità dell'utente (**adattivo**)
- Il processo delle tre fasi di progettazione, implementazione e test è quindi iterativo con fasi di durata molto più breve
- Extreme Programming, SCRUM, DSDM, Agile Modeling, Agile Data, Feature Driven Programming, Adaptive Software, Development di Highsmith, Use Cases, Use Stories sono alcune delle metodologie agili

Come affrontare il progetto o una sua fase

Approccio predittivo / deterministico

Deliverable rilasciati in modo completo al termine

Se la direzione individuata dal piano è errata, anche di poco, possono esserci scostamenti importanti tra obiettivo e risultato.



Approccio adattivo / agile

Incrementi di deliverable rilasciati al termine di ogni iterazione.

La direzione individuata dal piano viene adattata ad ogni iterazione per minimizzare gli scostamenti tra obiettivo e risultato

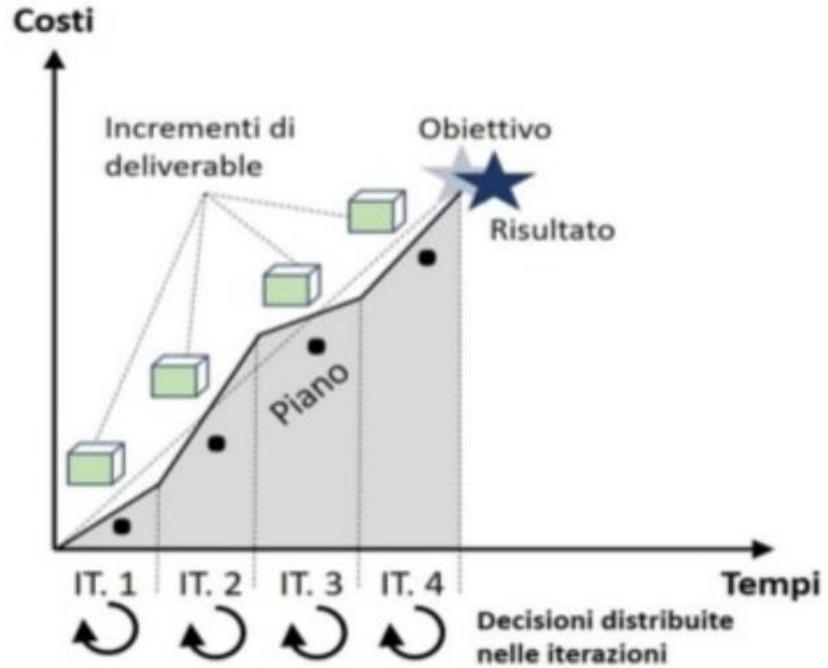


Immagine tratta dall'articolo di **Antonietta Fiorentino** *"Project Management e Agile, l'integrazione possibile"*
Pubblicato sul numero di settembre 2018 della rivista **Il Project Manager** . Editore **Franco Angeli**

Metodologie Agile

- Ricerca di una soluzione alla sempre maggiore complessità dei sistemi da progettare
- Manifesto per lo Sviluppo Agile di Software
  - http://agilemanifesto.org/
  - http://agilemanifesto.org/iso/it/manifesto.html
  - https://www.agilealliance.org/the-alliance

Manifesto per lo Sviluppo Agile di Software

Stiamo scoprendo modi migliori di creare software,
sviluppandolo e aiutando gli altri a fare lo stesso.
Grazie a questa attività siamo arrivati a considerare importanti:

Gli individui e le interazioni più che i processi e gli strumenti

Il software funzionante più che la documentazione esaustiva

La collaborazione col cliente più che la negoziazione dei contratti

Rispondere al cambiamento più che seguire un piano

Ovvero, fermo restando il valore delle voci a destra,
consideriamo più importanti le voci a sinistra.

Kent Beck
Mike Beedle
Arie van Bennekum
Alistair Cockburn
Ward Cunningham
Martin Fowler
James Grenning
Jim Highsmith
Andrew Hunt
Ron Jeffries
Jon Kern
Brian Marick
Robert C. Martin
Steve Mellor
Ken Schwaber
Jeff Sutherland
Dave Thomas

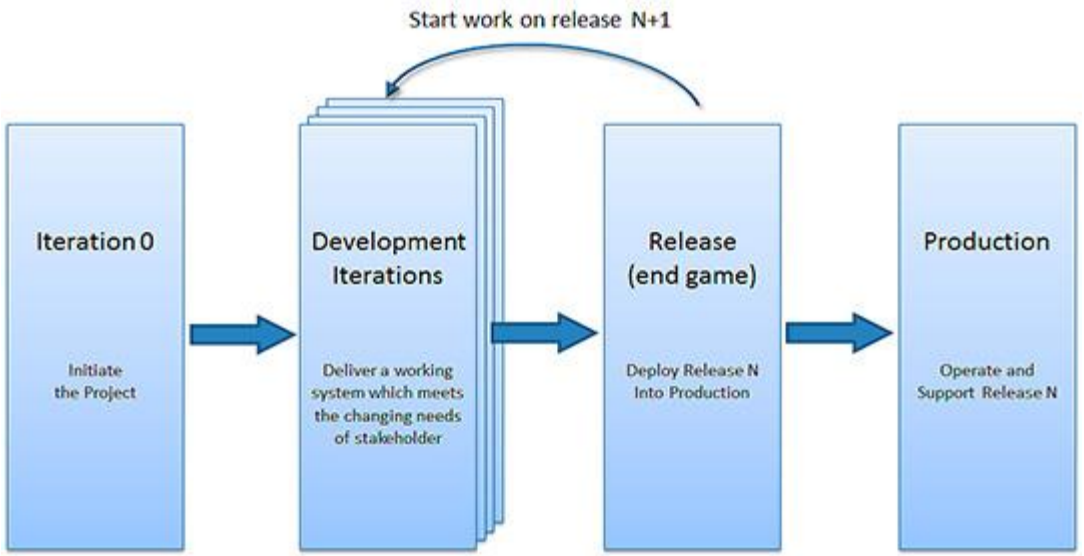
I Dodici Principi del Software Agile (1)

- 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
- **Consegniamo 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

I Dodici Principi del Software Agile (2)

- 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

Ciclo di vita dello sviluppo Agile

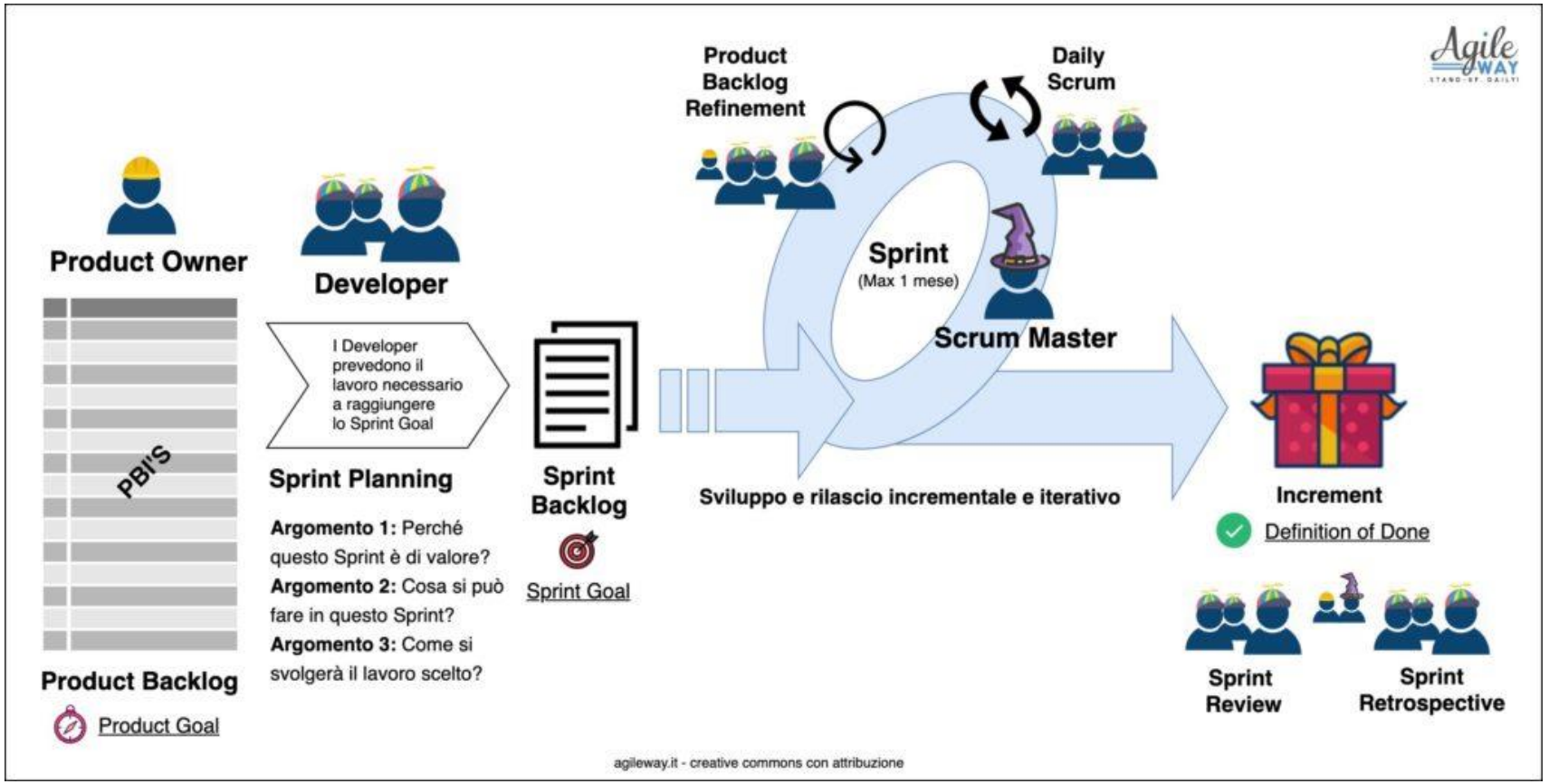


Ciclo di vita dello sviluppo Agile

- Prima fase denominata **Iteration 0** o **Cycle 0**
  - Ha lo scopo di lanciare il progetto: garantire una partecipazione attiva degli stakeholder, ottenere i supporti economici necessari, costituire il team, generare un modello iniziale dei requisiti e dell'architettura, predisporre gli ambienti
- Seconda fase caratterizzata da una **serie di iterazioni di sviluppo** in cui si realizza il sistema in maniera incrementale
  - Requisiti vengono analizzati utilizzando la tecnica del model storming con strumenti che rendono attiva la partecipazione del cliente, ad es. con use-case, diagrammi UML
  - Stretta collaborazione team-stakeholder, al termine di ogni ciclo sistema parzialmente funzionante mostrabile al cliente
- Terza fase costituita da **una o più iterazioni di rilascio** (o fine partita) relative alla Release N, al termine della quale si può procedere allo sviluppo della Release N+1
  - Test finale del sistema, deploy e formazione agli utenti
- Fase finale di produzione
  - Fase di esercizio del sistema

Scrum

Il termine Scrum è mutuato dal termine del rugby che indica il pacchetto di mischia ed è evidentemente una metafora del team di sviluppo che deve lavorare insieme in modo che tutti gli attori del progetto spingano nella stessa direzione, agendo come un'unica entità coordinata

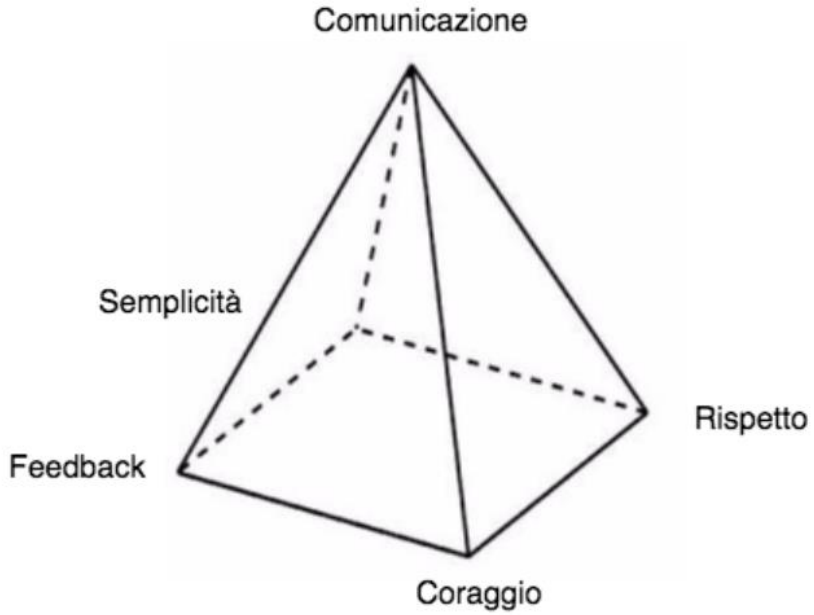


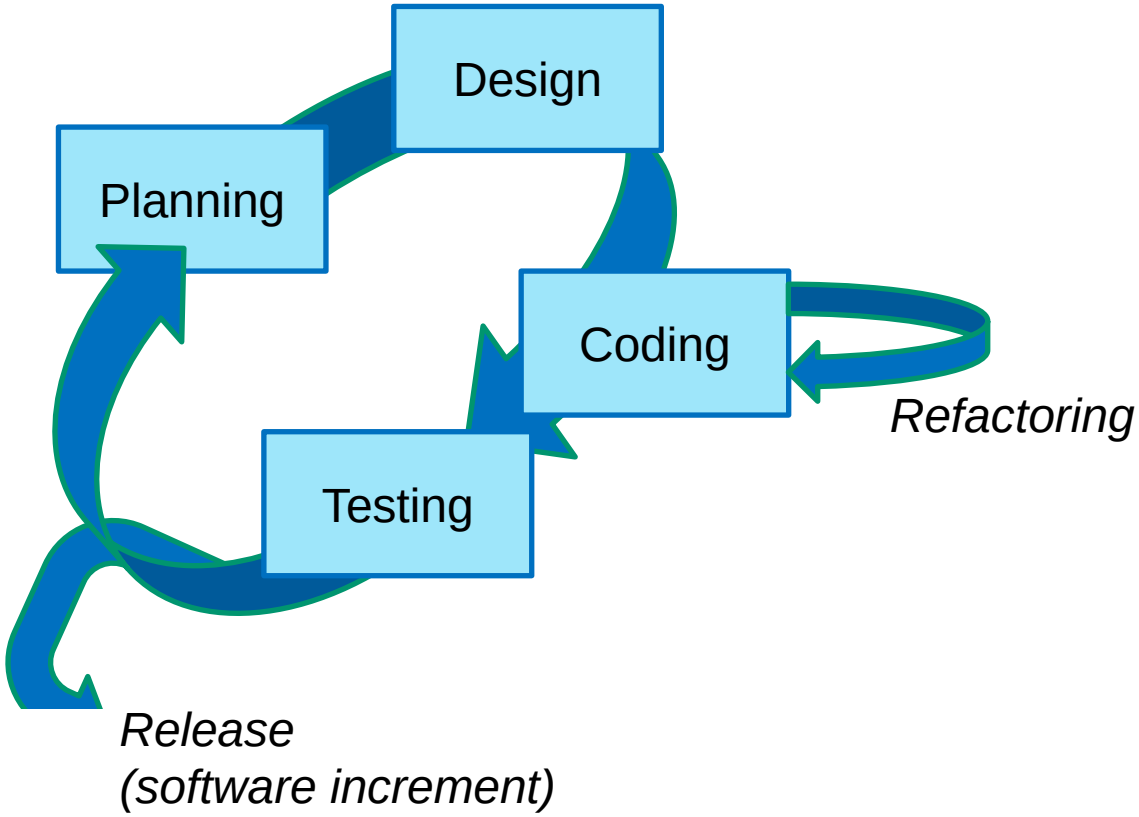
- Sprint: il team seleziona le storie degli utenti, le user story diventano quindi parte del backlog dello sprint, il team li suddivide in attività, attraverso iterazioni incrementali inizia a sviluppare le funzionalità
- Vengono organizzate riunioni giornaliere per discutere dei progressi e sollevare questioni
- Alla fine dello sprint, il software funzionante con le nuove funzionalità viene mostrato agli stakeholder per ottenere il loro feedback
- Il team conduce una retrospettiva sulla prestazione dello sprint e concorda le richieste di azione
- Il team acquisisce la sua velocità, che verrà utilizzata per pianificare gli sprint futuri
- Questo ciclo continua

https://www.agileway.it/scrum-metodologia-agile/

eXtreme Programming

Extreme Programming - valori





https://www.agileway.it/extreme-programming-metodologia-sviluppo-agile/

eXtreme Programming

- Extreme Programming (XP) è una metodologia agile e un approccio all'ingegneria del software formulato da Kent Beck (primo libro su XP nel 1999), Ward Cunningham e Ron Jeffries
- Linee guida
  - **Comunicazione**: tutti possono parlare con tutti, anche l'ultimo arrivato dei programmatori con il cliente
  - **Semplicità**: gli analisti mantengono la descrizione formale il più semplice e chiara possibile
  - **Feedback**: si testa il codice dal primo giorno
  - **Coraggio**: si dà in uso il sistema il prima possibile e si implementano i cambiamenti richiesti man mano
- Fasi
  - Pianificazione, Progettazione, Sviluppo, Testing

eXtreme Programming – fasi (1)

- **Pianificazione**
  - Inizia con la creazione di varie *user story*, viene assegnato un costo ad ogni user story, vengono raggruppate per determinare un risultato da consegnare in tempi brevi, viene stabilita la data di consegna, dopo il primo incremento la *project velocity* (misura dei completamenti di *user story* dell'interazione come stima dell'attività successiva) viene usata per definire le date delle consegne successive
- **Progettazione**
  - Segue il principio KISS (Keep It Simple, Stupid!)
  - Vengono utilizzate schede di Class Responsibility Collaboration (CRC): uno strumento usato per impostare un progetto software object-oriented attraverso un processo di brainstorming (ciascuna carta descrive una classe o un oggetto in modo sommario, indicando il nome della classe, le sue superclassi e sottoclassi se applicabile, le sue responsabilità, il nome di altre classi con cui questa classe collabora per svolgere i compiti di cui è responsabile, l'autore)
  - Per problemi di design difficoltosi si generano *spike solution*: prototipi operativi per il design di un aspetto specifico
  - Incoraggia il *refactoring*: un raffinamento iterativo del design interno del programma

eXtreme Programming – fasi (2)

- **Sviluppo**
  - Prevede la costruzione di unit test per le varie *user story* prima di iniziare a implementare il sistema
  - Incoraggia il *pair programming* (programmazione in coppia): è una tecnica agile di sviluppo del software nella quale due programmatori lavorano insieme a una postazione di lavoro. Uno dei due, indicato come *conducente* (driver) scrive il codice; l'altro, detto *osservatore* (observer) o *navigatore* (navigator), svolge un ruolo di supervisione e di revisione simultanea del codice
- **Test**
  - Tutti gli *unit test* vengono eseguiti quotidianamente
  - Test di accettazione (test funzionali) sono definiti dal cliente ed eseguiti per verificare le funzionalità visibili al cliente

Integrazione Continua / Consegna Continua (CI / CD)

CI / CD fa riferimento alle pratiche di ingegneria e distribuzione del software

- CI o integrazione continua è una pratica di ingegneria in cui i membri di un team di sviluppo integrano il loro codice a una frequenza molto elevata
- CI enfatizza l'uso di strumenti di automazione che eseguono build e test
- L'integrazione continua comporta anche il rilevamento di errori nel codice nelle prime fasi

CD o consegna continua è la pratica per garantire che il codice sia sempre in uno stato distribuibile

- Tutte le modifiche al codice (nuove funzionalità, correzioni di errori, esperimenti, modifiche alla configurazione) sono sempre pronte per la distribuzione in un ambiente di produzione
- Indifferentemente se la distribuzione coinvolge un sistema distribuito su larga scala, un sistema incorporato o un ambiente di produzione complesso

Per entrambe queste pratiche, è importante disporre di strumenti adeguati per il funzionamento della pipeline CI / CD

Le modifiche del software raggiungono la produzione più frequentemente, i clienti hanno più opportunità di sperimentare e fornire feedback

DevOps

DevOps come intersezione di Sviluppo (software engineering), technology operations e garanzia di qualità (Quality Assurance)

DevOps è una combinazione di sviluppo (Dev) e operazioni (Ops),
è l'unione di persone, processi e tecnologia
per offrire continuamente valore ai clienti

Vantaggi per i team?

DevOps permette a ruoli in precedenza isolati - sviluppo, operazioni IT, controllo della qualità e sicurezza – di coordinarsi e collaborare per fornire prodotti migliori e più affidabili

I team possono rispondere meglio alle esigenze dei clienti, migliorare l'attendibilità delle applicazioni create e raggiungere più rapidamente gli obiettivi

I team che adottano la cultura, le procedure e gli strumenti DevOps ottengono prestazioni elevate e creano più rapidamente prodotti, incrementando la soddisfazione dei clienti

- Accelerazione del time-to-market
- Adattamento al mercato e alla competizione
- Conservazione della stabilità e dell'affidabilità del sistema
- Miglioramento del tempo medio per il ripristino